

Fixed-Parameter and Approximation Algorithms for Maximum Agreement Forests of Multifurcating Trees

Chris Whidden, Robert G. Beiko, and Norbert Zeh*

May 3, 2013

Abstract

We present efficient algorithms for computing a maximum agreement forest (MAF) of a pair of multifurcating (nonbinary) rooted trees. Our algorithms match the running times of the currently best algorithms for the binary case. The size of an MAF corresponds to the subtree prune-and-regraft (SPR) distance of the two trees and is intimately connected to their hybridization number. These distance measures are essential tools for understanding reticulate evolution, such as lateral gene transfer, recombination, and hybridization. Multifurcating trees arise naturally as a result of statistical uncertainty in current tree construction methods.

1 Introduction

Phylogenetic trees are the standard model for representing the evolution of a set of species (taxa) through “vertical” inheritance [1]. Yet, genetic material can also be shared between contemporary organisms via lateral gene transfer, recombination or hybridization. These processes allow species to rapidly adapt to new environments as shown by, for example, the rapid spread of antibiotic resistance and other harmful traits in pathogenic bacteria [2]. Untangling vertical and lateral evolutionary histories is thus both difficult and of great importance. To do so often requires the comparison of phylogenetic trees for individual gene histories with a reference tree. Distance measures that model reticulation events using subtree prune-and-regraft (SPR) [1] and hybridization [3] operations are of particular interest in such comparisons due to their direct evolutionary interpretations [3, 4].

These distance measures are biologically meaningful but also NP-hard to compute [5–7]. As a result, there has been significant effort to develop efficient fixed-parameter [8–12] and approximation [8, 13, 14] algorithms to compute these distances, most of which use the equivalent notion of maximum agreement forests (MAFs) [3, 5, 15]. Efficient algorithms for computing these distances have generally been restricted to binary trees. The exceptions are reduction rules for computing hybridization numbers of nonbinary trees [16] and a recent depth-bounded search algorithm [17] for computing the subtree prune-and-regraft distance of such trees.

Multifurcations (or *polytomies*) are vertices of a tree with two or more children. A multifurcation is *hard* if it indeed represents an inferred common ancestor which produced three or more species as direct descendants; it is *soft* if it simply represents ambiguous evolutionary relationships [18]. Simultaneous speciation events are assumed to be rare, so a common assumption is that all multifurcations are soft. If we force the resolution of multifurcating trees into binary trees, then we infer evolutionary relationships that are not supported by the original data and may infer

*Faculty of Computer Science, Dalhousie University, Halifax, Nova Scotia, Canada.
E-mail: {whidden, beiko, nzech}@cs.dal.ca

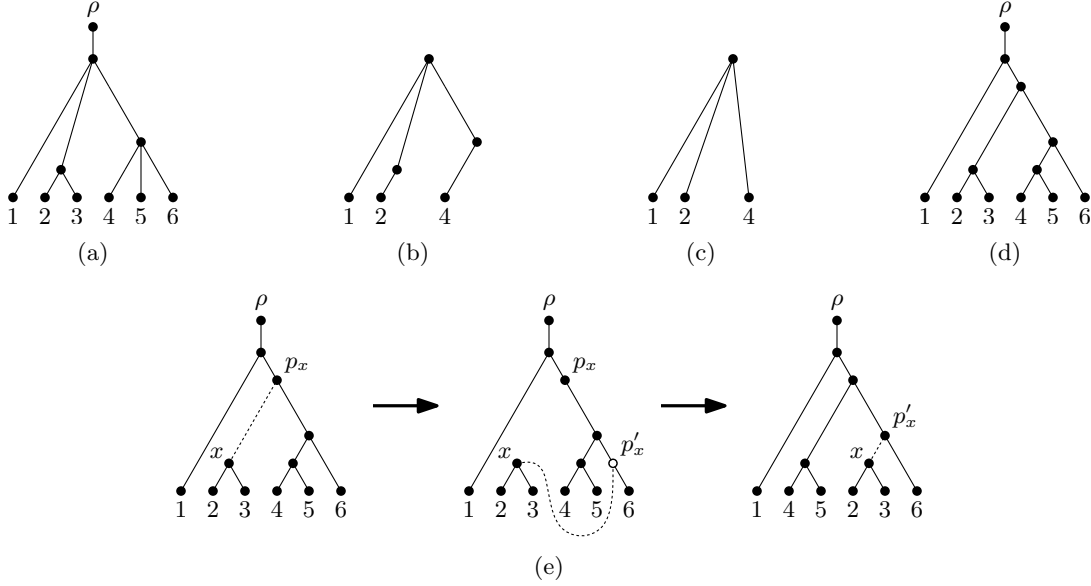


Figure 1: (a) An X -tree T . (b) The subtree $T(V)$ for $V = \{1, 2, 4\}$. (c) $T|V$. (d) A binary resolution of T . (e) Illustration of an SPR operation applied to the binary resolution of T .

meaningless reticulation events. Thus, it is crucial to develop efficient algorithms to compare multifurcating trees directly.

In this paper, we extend the fastest approximation and fixed-parameter algorithms for computing MAFs of binary rooted trees to multifurcating trees (thus showing that computing MAFs for multifurcating trees is fixed-parameter tractable). The size of an MAF of two binary trees is equivalent to their SPR distance. In keeping with the assumption that multifurcations are soft, we define an MAF of two multifurcating trees so that its size is equivalent to what we call the *soft* SPR distance: the minimum number of SPR operations required to transform a binary resolution of one tree into a binary resolution of the other. This distinction avoids the inference of meaningless differences between the trees that arises, for example, when one tree has a set of resolved bifurcations that are part of a multifurcation in the other tree. This is similar to the extension of the hybridization number to multifurcating trees by Linz and Semple [16].

Our fixed-parameter algorithm achieves the same running time as in the binary case. Our approximation algorithm achieves the same approximation factor as in the binary case at the cost of increasing the running time from linear to $O(n \lg n)$. These results are not trivial extensions of the algorithm for binary trees. They require new structural insights and a novel method for terminating search branches of the depth-bounded search tree, coupled with a careful analysis of the resulting recurrence relation.

The rest of this paper is organized as follows. Section 2 introduces the necessary terminology and notation. Section 3 presents the key structural results for multifurcating agreement forests. Section 4 presents our new algorithms based on these results. Finally, in Section 5, we present closing remarks and discuss open problems and possible extensions of this work.

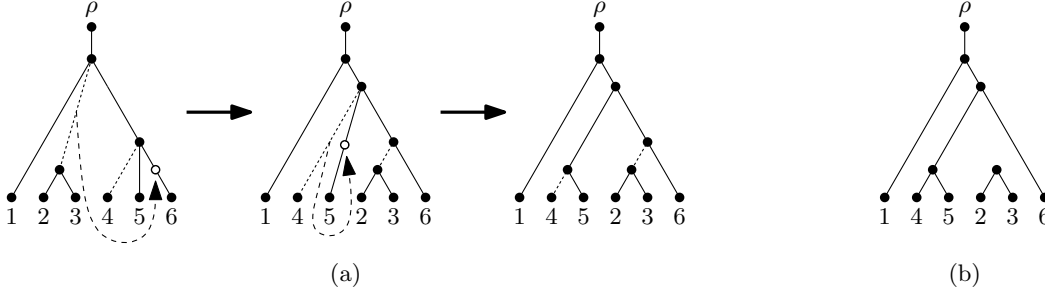


Figure 2: (a) SPR operations transforming the tree T from Figure 1(a) into the second tree in Figure 1(e). Each operation changes the top endpoint of one of the dotted edges. The hard SPR distance between the two trees is 2. (b) The MAF representing only the first transfer of (a) (equivalently, Figure 1(e)). The second transfer is unnecessary if the multifurcation represents ambiguous data rather than simultaneous speciation. Thus, the soft SPR distance is 1.

2 Preliminaries

Throughout this paper, we mostly use the definitions and notation from [5, 13, 14, 16, 19, 20]. A (*rooted phylogenetic*) X -tree is a rooted tree T whose leaves are the elements of a label set X and whose non-root internal nodes have at least two children each; see Figure 1(a). T is *binary* (or *bifurcating*) if all internal nodes have exactly two children each, otherwise it is *multifurcating*. The root of T has label ρ and has one child. Throughout this paper, we consider ρ to be a member of X . For a subset V of X , $T(V)$ is the smallest subtree of T that connects all nodes in V ; see Figure 1(b). The V -tree induced by T is the smallest tree $T|V$ that can be obtained from $T(V)$ by *contracting* unlabelled nodes with only one child, that is, by merging each such node with one of its neighbours and removing the edge between them. See Figure 1(c). An *expansion* does the opposite: It splits a node v into two nodes v_1 and v_2 such that v_1 is v_2 's parent and divides the children of v into two subsets that become the children of v_1 and v_2 , respectively. For brevity, we refer to this operation as expanding the subset of v 's children that become v_2 's children.

Let T_1 and T_2 be two X -trees. We say that T_2 *resolves* T_1 or, equivalently, T_2 is a *resolution* of T_1 if T_1 can be obtained from T_2 by contracting internal edges. T_2 is a *binary resolution* of T_1 if T_2 is binary. See Figure 1(d).

A *subtree prune-and-regraft* (SPR) operation on a binary rooted X -tree T cuts an edge xp_x , where p_x denotes the parent of x . This divides T into subtrees T_x and T_{p_x} containing x and p_x , respectively. Then it introduces a node p'_x into T_{p_x} by subdividing an edge of T_{p_x} and adds an edge xp'_x , thereby making x a child of p'_x . Finally, p_x is removed using a contraction. See Figure 1(e). On a multifurcating tree, an SPR operation may also use any existing node of T_{p_x} as p'_x and contracts p_x only if it has only one child besides x .

SPR operations give rise to a distance measure $d_{SPR}(\cdot, \cdot)$ between binary X -trees, defined as the minimum number of such operations required to transform one tree into the other. The trees in Figure 1(e), for example, have SPR distance $d_{SPR}(T_1, T_2) = 1$. An analogous distance measure, which we call the *hard SPR distance*, could be defined for multifurcating X -trees; however, under the assumption that most multifurcations are soft, this would capture differences between the trees that are meaningless. Instead, we define the *soft SPR distance* $d_{sSPR}(T_1, T_2)$ between two multifurcating trees T_1 and T_2 to be the minimum SPR distance of all pairs of binary resolutions of T_1 and T_2 .¹ For simplicity, we simply refer to this as the SPR distance in the remainder of this paper. These

¹This is similar to the generalization of the hybridization number used by Linz and Semple [16]

two distance measures are illustrated in Figure 2. Note that the soft SPR distance is not a metric but captures the minimum number of SPR operations needed to explain the difference between the two trees.

These distance measures are related to the sizes of appropriately defined agreement forests. To define these, we first introduce some terminology. For a forest F whose components T_1, T_2, \dots, T_k have label sets X_1, X_2, \dots, X_k , we say F *yields* the forest with components $T_1|X_1, T_2|X_2, \dots, T_k|X_k$; if $X_i = \emptyset$, then $T_i(X_i) = \emptyset$ and, hence, $T_i|X_i = \emptyset$. For a subset E of edges of F , we use $F - E$ to denote the forest obtained by deleting the edges in E from F , and $F \div E$ to denote the forest yielded by $F - E$. Thus, $F \div E$ is the contracted form of $F - E$. We say $F \div E$ is a *forest of* F .

Given X -trees T_1 and T_2 and forests F_1 of T_1 and F_2 of T_2 , a forest F is an *agreement forest* (AF) of F_1 and F_2 if it is a forest of a binary resolution of F_1 and of a binary resolution of F_2 . F is a *maximum agreement forest* (MAF) of F_1 and F_2 if there is no AF of F_1 and F_2 with fewer components. An MAF of the trees from Figure 2(a) is shown in Figure 2(b). We denote the number of components in an MAF of F_1 and F_2 by $m(F_1, F_2)$, and the size of the smallest edge set E such that $F' \div E$ is an AF of F_1 and F_2 by $e(F_1, F_2, F)$, where F is a forest of F_2 and F' is a binary resolution of F . Bordewich and Semple [5] showed that, for two *binary* rooted X -trees T_1 and T_2 , $d_{SPR}(T_1, T_2) = e(T_1, T_2, T_2) = m(T_1, T_2) - 1$. This implies that $d_{sSPR}(T_1, T_2) = e(T_1, T_2, T_2) = m(T_1, T_2) - 1$ for two arbitrary rooted X -trees because $d_{sSPR}(T_1, T_2)$, $e(T_1, T_2, T_2)$, and $m(T_1, T_2)$ are taken as the minimum over all binary resolutions of T_1 and T_2 . Thus, to determine the SPR distance between two rooted X -trees, we need to compute a binary MAF of the two trees.

We write $a \sim_F b$ when there exists a path between two nodes a and b of a forest F . For a node x of F , F^x denotes the subtree of F induced by all descendants of x , inclusive. For two rooted forests F_1 and F_2 and a node $a \in F_1$, we say that a *exists* in F_2 if there exists a node a' in F_2 such that $F_1^a = F_2^{a'}$. For simplicity, we refer to both a' simply as a . For forests F_1 and F_2 and nodes $a, c \in F_1$ with a common parent, we say $\{a, c\}$ is a *sibling pair* of F_1 if a and c exist in F_2 . Figure 3 shows such a sibling pair. We say $\{a_1, a_2, \dots, a_m\}$ is a *sibling group* if $\{a_i, a_j\}$ is a sibling pair of F_1 , for all $1 \leq i < j \leq m$, and a_1 has no sibling not in the group.

The correctness proofs of our algorithms in the next sections make use of the following three lemmas. Lemma 1 was shown by Bordewich et al. [14] for binary trees. The proof trivially extends to multifurcating trees.

Lemma 1. *Let F be a forest of an X -tree, e and f edges of F , and E a subset of edges of F such that $f \in E$ and $e \notin E$. Let v_f be the end vertex of f closest to e , and v_e an end vertex of e . If (1) $v_f \sim_{F-E} v_e$ and (2) $x \sim_{F-(E \cup \{e\})} v_f$, for all $x \in X$, then $F \div E = F \div (E \setminus \{f\} \cup \{e\})$.*

Let F_1 and F_2 be forests of X -trees T_1 and T_2 , respectively. Any agreement forest of F_1 and F_2 is clearly also an agreement forest of T_1 and T_2 . Conversely, an agreement forest of T_1 and T_2 is an agreement forest of F_1 and F_2 if it is a forest of F_2 and there are no two leaves a and b such that $a \sim_{F_2} b$ but $a \not\sim_{F_1} b$. This is formalized in the following lemma. Our algorithms ensure that any intermediate forests F_1 and F_2 they produce have this latter property. Thus, this lemma allows us to reason about agreement forests of F_1 and F_2 and of T_1 and T_2 interchangeably, as long as they are forests of F_2 .

Lemma 2. *Let F_1 and F_2 be forests of X -trees T_1 and T_2 , respectively. Let \dot{F}_1 be the union of trees $\dot{T}_1, \dot{T}_2, \dots, \dot{T}_k$ and \dot{F}_2 be the union of forests $\dot{F}_1, \dot{F}_2, \dots, \dot{F}_k$ such that \dot{T}_i and \dot{F}_i have the same label set, for all $1 \leq i \leq k$. Let \dot{F}_2' be a resolution of \dot{F}_2 . $\dot{F}_2' \div E$ is an AF of T_1 and T_2 if and only if it is an AF of F_1 and F_2 .*

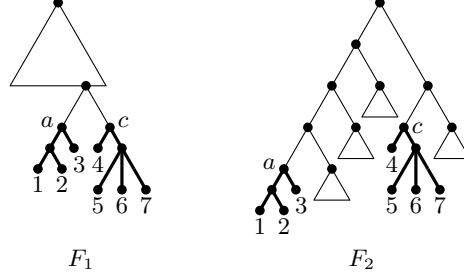


Figure 3: A sibling pair $\{a, c\}$ of two forests F_1 and F_2 : a and c have a common parent in F_1 , and both subtrees F_1^a and F_1^c exist also in F_2 .

To use Lemma 1 to prove structural properties of agreement forests, which are defined in terms of resolutions of forests, we also need the following lemma, which specifies when an expansion does not change the SPR distance. Its proof is provided in Section S6.1 in the supplementary material.

Lemma 3. *Let F_1 and F_2 be resolutions of forests of rooted X -trees T_1 and T_2 , and let $F \div E$ be a maximum agreement forest of F_1 and F_2 , where F is a binary resolution of F_2 . Let $a_1, a_2, \dots, a_p, a_{p+1}, \dots, a_m$ be the children of a node in F_2 and let F'_2 be the result of expanding $\{a_{p+1}, a_{p+2}, \dots, a_m\}$ in F_2 . If $a'_i \approx_{F \div E} a'_j$, for all $1 \leq i \leq p$, $p+1 \leq j \leq m$, and all leaves $a'_i \in F_2^{a_i}$ and $a'_j \in F_2^{a_j}$, then $e(F_1, F_2, F_2) = e(F_1, F_2, F'_2)$.*

A triple $ab|c$ of a rooted forest F is defined by a set $\{a, b, c\}$ of three leaves in the same component of F and such that the path from a to b in F is disjoint from the path from c to the root of the component. Multifurcating trees also allow for triples $a|b|c$ where a , b , and c share the same lowest common ancestor (LCA). A triple $ab|c$ of a forest F_1 is *compatible* with a forest F_2 if it is also a triple of F_2 or F_2 contains the triple $a|b|c$; otherwise it is *incompatible* with F_2 .

An agreement forest of two forests F_1 and F_2 cannot contain a triple incompatible with either of the two forests. Thus, we have the following observation.

Observation 1. *Let F_1 and F_2 be forests of rooted X -trees T_1 and T_2 , and let F be an agreement forest of F_1 and F_2 . If $ab|c$ is a triple of F_1 incompatible with F_2 , then $a \approx_F b$ or $a \approx_F c$.*

For two forests F_1 and F_2 with the same label set, two components C_1 and C_2 of F_2 are said to *overlap* in F_1 if there exist leaves $a, b \in C_1$ and $c, d \in C_2$ such that the paths from a to b and from c to d in F_1 exist and are not edge-disjoint. The following lemma is an easy extension of a lemma of [14], which states the same result for binary trees instead of binary forests.

Lemma 4. *Let F_1 and F_2 be binary resolutions of forests of two X -trees T_1 and T_2 , and denote the label sets of the components of F_1 by X_1, X_2, \dots, X_k and the label sets of the components of F_2 by Y_1, Y_2, \dots, Y_l . F_2 is a forest of F_1 if and only if (1) for every Y_j , there exists an X_i such that $Y_j \subseteq X_i$, (2) no two components of F_2 overlap in F_1 , and (3) no triple of F_2 is incompatible with F_1 .*

3 The Structure of Multifurcating Agreement Forests

This section presents the structural results that provide the intuition and formal basis for the algorithms presented in Section 4. All these algorithms start with a pair of trees (T_1, T_2) and

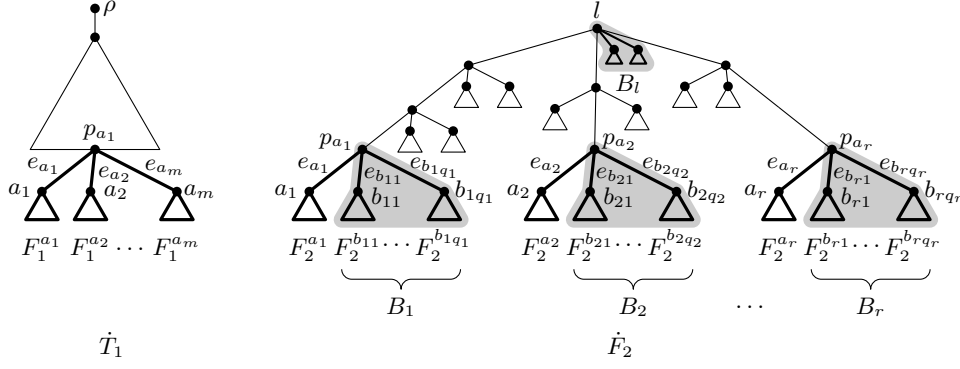


Figure 4: Tree labels for a sibling group $\{a_1, a_2, \dots, a_m\}$ such that a_1, a_2, \dots, a_r share a minimal LCA l .

then cut edges, expand sets of nodes, remove agreeing components from consideration, and merge sibling pairs until the resulting forests are identical. The intermediate state is that T_1 and T_2 have been resolved and reduced to forests F_1 and F_2 , respectively. F_1 consists of a tree \hat{T}_1 and a set of components F_0 that exist in F_2 . F_2 has two sets of components. One is F_0 . The other, \hat{F}_2 , has the same label set as \hat{T}_1 but may not agree with \hat{T}_1 . The key in each iteration is deciding which edges in \hat{F}_2 to cut next or which nodes to expand, in order to make progress towards an MAF of T_1 and T_2 . The results in this section identify small edge sets in \hat{F}_2 such that at least one edge in each of these sets has the property that cutting it reduces $e(T_1, T_2, F_2)$ by one. Some of these edges are introduced by expanding nodes. The approximation algorithm cuts all edges in the identified set, and the size of the set gives the approximation ratio of the algorithm. The FPT algorithm tries each edge in the set in turn, so that the size of the set gives the branching factor for a depth-bounded search algorithm.

Let $\{a_1, a_2, \dots, a_m\}$ be a sibling group of \hat{T}_1 . If there exist indices $i \neq j$ such that a_i and a_j are also siblings in F_2 , we can expand this sibling pair $\{a_i, a_j\}$ and replace a_i and a_j with their parent node (a_i, a_j) in the sibling group. If there exists an index i such that $F_2^{a_i}$ is a component of F_2 , then we can cut a_i 's parent edge in F_1 , thereby removing a_i from the sibling group. Thus, we can assume a_i and a_j are not siblings in F_2 , for all $1 \leq i < j \leq m$, and $F_2^{a_i}$ is not a component of F_2 , for all $1 \leq i \leq m$. We have $a_i \in \hat{F}_2$, for all $1 \leq i \leq m$, because \hat{T}_1 and \hat{F}_2 have the same label set. Let $B_i = \{b_{i1}, b_{i2}, \dots, b_{iq_i}\}$ be the siblings of a_i in F_2 , for $1 \leq i \leq m$. We use e_x to denote the edge connecting a node x to its parent p_x , e_{B_i} to denote the edge introduced by expanding B_i , and p_{B_i} to denote the common parent of the nodes in B_i . $F_2 - \{e_{B_i}\}$ denotes the forest obtained from F_2 by expanding B_i and then cutting e_{B_i} , and we use $F_2^{B_i}$ to denote the subforest of F_2 comprised of the subtrees $F_2^{b_{i1}}, F_2^{b_{i2}}, \dots, F_2^{b_{iq_i}}$.

Consider a subset $\{a_{i_1}, a_{i_2}, \dots, a_{i_r}\}$ of a sibling group $\{a_1, a_2, \dots, a_m\}$. We say $a_{i_1}, a_{i_2}, \dots, a_{i_r}$ share their LCA l if $l = LCA_{F_2}(a_i, a_j)$, for all $i, j \in \{i_1, i_2, \dots, i_r\}$, $i \neq j$. If, in addition, $LCA_{F_2}(a_i, a_j)$ is not a proper descendant of l , for all $1 \leq i < j \leq m$, we say that $a_{i_1}, a_{i_2}, \dots, a_{i_r}$ share a minimal LCA l . For simplicity, we always order the elements of the group so that $\{a_{i_1}, a_{i_2}, \dots, a_{i_r}\} = \{a_1, a_2, \dots, a_r\}$ and assume the subset that shares l is maximal, that is, a_i is not a descendant of l , for all $r < i \leq m$. We use B_l to denote the set of children of l that do not have any member a_i of the sibling group as a descendant. Note that $B_l \subseteq B_i$ when a_i is a child of l . These labels are illustrated in Figure 4.

Our first result shows that at least one of the edges e_{a_1} , e_{a_2} , e_{B_1} , and e_{B_2} has the property that cutting it reduces $e(T_1, T_2, F_2)$ by one. This implies that cutting e_{a_1} , e_{a_2} , $e_{p_{a_1}}$, and $e_{p_{a_2}}$ reduces

$e(T_1, T_2, F_2)$ by at least 1.

Theorem 1. *Let F_1 and F_2 be forests of rooted X -trees T_1 and T_2 , respectively, and assume F_1 consists of a tree \dot{T}_1 and a set of components that exist in F_2 . Let $\{a_1, a_2, \dots, a_m\}$ be a sibling group of \dot{T}_1 such that either a_1, a_2, \dots, a_r share a minimal LCA l in F_2 or a_2, a_3, \dots, a_r share a minimal LCA l in F_2 and $a_1 \approx_{F_2} a_i$, for all $2 \leq i \leq m$; a_1 is not a child of l ; a_2 is not a child of l unless $r = 2$; a_i and a_j are not siblings in F_2 , for all $1 \leq i < j \leq m$; and $F_2^{a_i}$ is not a component of F_2 , for all $1 \leq i \leq m$. Then*

(i) $e(T_1, T_2, F_2 - \{e_x\}) = e(T_1, T_2, F_2) - 1$, for some $x \in \{a_1, a_2, B_1, B_2\}$.

(ii) $e(T_1, T_2, F_2 - \{e_{a_1}, e_{a_2}, e_{p_{a_1}}, e_{p_{a_2}}\}) \leq e(T_1, T_2, F_2) - 1$.

Proof. (ii) follows immediately from (i) because cutting $\{e_{a_1}, e_{a_2}, e_{p_{a_1}}, e_{p_{a_2}}\}$ is equivalent to cutting $\{e_{a_1}, e_{a_2}, e_{B_1}, e_{B_2}\}$. For (i), it suffices to prove that there exist a binary resolution F of F_2 and an edge set E of size $e(T_1, T_2, F_2)$ such that $F \div E$ is an MAF of T_1 and T_2 and $E \cap \{e_{a_1}, e_{a_2}, e_{B_1}, e_{B_2}\} \neq \emptyset$.

So assume $F \div E$ is an MAF of T_1 and T_2 and $E \cap \{e_{a_1}, e_{a_2}, e_{B_1}, e_{B_2}\} = \emptyset$. By Lemma 2, $F \div E$ is also an MAF of F_1 and F_2 . We prove that we can replace some edge $f \in E$ with an edge in $\{e_{a_1}, e_{a_2}, e_{B_1}, e_{B_2}\}$ without changing $F \div E$.

First assume $b'_1 \approx_{F \div E} x$, for all leaves $b'_1 \in F_2^{B_1}$ and $x \notin F_2^{B_1}$. By Lemma 3, expanding B_1 does not change $e(F_1, F_2, F_2)$, so we can assume F contains this expansion.² Now we choose an arbitrary leaf $b'_1 \in B_1$ and the first edge $f \in E$ on the path from p_{B_1} to b'_1 . By Lemma 1, $F \div E = F \div (E \setminus \{f\} \cup \{e_{B_1}\})$. If $b'_2 \approx_{F \div E} x$, for all leaves $b'_2 \in F_2^{B_2}$ and $x \notin F_2^{B_2}$, $a'_1 \approx_{F \div E} p_{a_1}$, for all leaves $a'_1 \in F_2^{a_1}$, or $a'_2 \approx_{F \div E} p_{a_2}$, for all leaves $a'_2 \in F_2^{a_2}$, then the same argument shows that $F \div E = F \div (E \setminus \{f\} \cup \{e_x\})$, for $x = B_2$, $x = a_1$, and $x = a_2$, respectively. Thus, we can assume there exist leaves $a'_1 \in F_2^{a_1}$, $a'_2 \in F_2^{a_2}$, $b'_1 \in F_2^{B_1}$, and $b'_2 \in F_2^{B_2}$ such that $a'_1 \sim_{F \div E} p_{a_1} \sim_{F \div E} b'_1$ and $a'_2 \sim_{F \div E} p_{a_2} \sim_{F \div E} b'_2$.

Now recall that either a_1, a_2, \dots, a_r share the minimal LCA l and a_1 is not a child of l or a_2, a_3, \dots, a_r share the minimal LCA l and $a_1 \approx_{F_2} a_i$, for all $2 \leq i \leq m$. In either case, $a_i \notin F_2^{B_1}$, for all $1 \leq i \leq m$. Since a_i also is not an ancestor of p_{B_1} , this shows that $b'_1 \notin F_2^{a_i}$, for all $1 \leq i \leq m$. Thus, F_1 contains the triple $a'_1 a'_2 | b'_1$, while F_2 contains the triple $a'_1 b'_1 | a'_2$ or $a'_1 \approx_{F_2} a'_2$. By Observation 1, this implies that $a'_1 \sim_{F \div E} p_{a_1} \sim_{F \div E} b'_1 \approx_{F \div E} a'_2 \sim_{F \div E} p_{a_2} \sim_{F \div E} b'_2$ and, hence, $b'_2 \notin F_2^{a_1}$. Since a_2 is not a child of l unless $r = 2$, we also have $b'_2 \notin F_2^{a_i}$, for all $2 \leq i \leq m$. Thus, F_1 also contains the triple $a'_1 a'_2 | b'_2$, which implies that the components of $F \div E$ containing a'_1, b'_1 and a'_2, b'_2 overlap in F_1 , a contradiction. \square

Theorem 1 covers every case where some minimal LCA l exists. If there is no such minimal LCA, then each a_i must be in a separate component of F_2 . In the following lemma we show the stronger result that cutting e_{a_1} or e_{a_2} reduces $e(T_1, T_2, F_2)$ by one in this case (which immediately implies that claims (i) and (ii) of Theorem 1 also hold in this case).

Lemma 5 (Isolated Siblings). *If $a_1 \approx_{F_2} a_i$, for all $i \neq 1$, $a_2 \approx_{F_2} a_j$, for all $j \neq 2$, and $F_2^{a_i}$ is not a component of F_2 , for all $1 \leq i \leq m$, then there exist a resolution F of F_2 and an edge set E of size $e(T_1, T_2, F_2)$ such that $F \div E$ is an AF of T_1 and T_2 and $E \cap \{e_{a_1}, e_{a_2}\} \neq \emptyset$.*

Proof. Consider an edge set E of size $e(T_1, T_2, F_2)$ and such that $F \div E$ is an AF of F_1 and F_2 , and assume E is chosen so that $|E \cap \{e_{a_1}, e_{a_2}\}|$ is maximized. Assume for the sake of contradiction that

²In fact, using the same ideas as in the proof of Lemma 3, it is not difficult to see that this expansion never precludes obtaining the same forest $F \div E$ by cutting a different set of $|E|$ edges. We discuss the importance of this to hybridization and reticulate analysis in Section 5.

$E \cap \{e_{a_1}, e_{a_2}\} = \emptyset$. Then, by the same arguments as in the proof of Theorem 1, there exist leaves $a'_1 \in F_2^{a_1}$ and $a'_2 \in F_2^{a_2}$ such that $a'_1 \sim_{F \div E} a_1$ and $a'_2 \sim_{F \div E} a_2$. Since $\{a_1, a_2, \dots, a_m\}$ is a sibling group of F_1 but $a_1 \not\sim_{F_2} a_i$, for all $i \neq 1$, and $a_2 \not\sim_{F_2} a_j$, for all $j \neq 2$, we must have $a'_1 \not\sim_{F-E} x$, for all leaves $x \notin F_2^{a_1}$, or $a'_2 \not\sim_{F-E} x$, for all leaves $x \notin F_2^{a_2}$. W.l.o.g. assume the former. Since $F_2^{a_1}$ is not a component of F_2 , there exists a leaf $y \notin F_2^{a_1}$ such that $a_1 \sim_{F_2} y$ and, hence, $a'_1 \sim_{F_2} y$. For each such leaf y , the path from a'_1 to y in F contains an edge in E because $a'_1 \not\sim_{F \div E} y$, and this edge does not belong to $F_2^{a_1}$ because $a'_1 \sim_{F \div E} a_1$. We pick an arbitrary such leaf y , and let f be the first edge in E on the path from a'_1 to y . The edges e_{a_1} and f satisfy the conditions of Lemma 1, that is, $F \div E = F \div (E \setminus \{f\} \cup \{e_{a_1}\})$. This contradicts the choice of E . \square

Theorem 1 and Lemma 5 are all that is needed to obtain a linear-time 4-approximation algorithm and an FPT algorithm with running time $O(4^k n)$ for computing rooted MAFs, an observation made independently in [17]. To improve on this in our algorithms in Section 4, we exploit a useful observation from the proof of Theorem 1: if there exists an MAF $F \div E$ and leaves $a'_1 \in F_2^{a_1}$ and $b'_1 \in F_2^{B_1}$ such that $a'_1 \sim_{F \div E} b'_1$, then $a'_2 \not\sim_{F \div E} b'_2$, for all $a'_2 \in F_2^{a_2}$ and $b'_2 \in F_2^{B_2}$. This implies that, if we choose to cut e_{a_2} or e_{B_2} and keep both e_{a_1} and e_{B_1} in a branch of our FPT algorithm, then we need only decide which edge, e_{a_j} or e_{B_j} , to cut in each pair $\{e_{a_j}, e_{B_j}\}$, for $3 \leq j \leq r$, in subsequent steps of this branch. This allows us to follow each 4-way branch in the algorithm (where we decide whether to cut e_{a_1} , e_{B_1} , e_{a_2} or e_{B_2}) by a series of 2-way branches. We cannot use this idea when a sibling group consists of only two nodes. The following lemma addresses this case. Its proof is provided in Section S6.2 of the supplementary material.

Lemma 6. *Let T_1 and T_2 be rooted X -trees, and let F_1 be a forest of T_1 and F_2 a forest of T_2 . Suppose F_1 consists of a tree \hat{T}_1 and a set of components that exist in F_2 . Let $\{a_1, a_2\}$ be a sibling group of \hat{T}_1 such that neither $F_2^{a_1}$ nor $F_2^{a_2}$ is a component of F_2 and, if a_1 and a_2 share a minimal LCA l , then a_1 is not a child of l . In particular, a_1 and a_2 are not siblings in F_2 . Then*

- (i) $e(T_1, T_2, F_2 - \{e_x\}) = e(T_1, T_2, F_2) - 1$, for some $x \in \{a_1, a_2, B_1\}$.
- (ii) $e(T_1, T_2, F_2 - \{e_{a_1}, e_{a_2}, e_{p_{a_1}}\}) \leq e(T_1, T_2, F_2) - 1$.

Next we examine the structure of a sibling group more closely as a basis for a refined analysis that leads to our final FPT algorithm with running time $O(2.42^k n)$. First we require the notion of *pendant subtrees* that we will be able to cut in unison. Let a_1, a_2, \dots, a_r be the members of a sibling group $\{a_1, a_2, \dots, a_m\}$ that share a minimal LCA l in F_2 , and consider the path from a_i to l , for any $1 \leq i \leq r$. Let x_1, x_2, \dots, x_{s_i} be the nodes on this path, excluding a_i and l . For each x_j , let B_{ij} be the set of children of x_j , excluding the child that is an ancestor of a_i , and let $F_2^{B_{ij}}$ be the subforest of F_2 consisting of all subtrees F_2^b , $b \in B_{ij}$. Note that $B_{i1} = B_i$, and $F_2^{B_{i1}} = F_2^{B_i}$, if $s_i > 0$. Analogously to the definition of e_{B_i} , we use $e_{B_{ij}}$, for $1 \leq j \leq s_i$, to denote the edge introduced by expanding the nodes in B_{ij} in F_2 and $F_2 - \{e_{B_{ij}}\}$ to denote the forest obtained by expanding B_{ij} and cutting edge $e_{B_{ij}}$. Note that expanding B_{ij} turns the forest $F_2^{B_{ij}}$ into a single *pendant subtree* attached to x_j . We distinguish five cases for the structure of the subtree of F_2 induced by the paths between a_1, a_2, \dots, a_r and l :

Isolated Siblings: $a_1 \not\sim_{F_2} a_i$, for all $i \neq 1$, and $a_2 \not\sim_{F_2} a_j$, for all $j \neq 2$.

At Most One Pendant Subtree: a_1, a_2, \dots, a_r share a minimal LCA l in F_2 , $s_i = 1$, for all $1 \leq i < r$, and a_r is a child of l .

One Pendant Subtree: a_1, a_2, \dots, a_r share a minimal LCA l in F_2 and $s_i = 1$, for all $1 \leq i \leq r$.

Multiple Pendant Subtrees, $m = 2$: a_1 and a_2 share a minimal LCA l in F_2 and $s_1 + s_2 \geq 2$.

Multiple Pendant Subtrees, $m > 2$: a_1, a_2, \dots, a_r share a minimal LCA l in F_2 and $s_1 \geq 2$.

Since we assume $F_2^{a_i}$ is not a component of F_2 , for all $1 \leq i \leq m$, and no two nodes a_i and a_j in the sibling group are siblings in F_2 , we need only consider cases where at most one a_i is a child of some minimal LCA l , and we always label it a_r . Hence, $s_i > 0$, for all $i < r$ such that $a_i \sim_{F_2} l$. Thus, the five cases above cover every possible configuration of a sibling group where we must cut an edge of F_2 .

The following four lemmas provide stronger statements than Theorem 1 about subsets of edges of a resolution F of F_2 that need to be cut in each of the last four cases above in order to make progress towards an AF of T_1 and T_2 . All four lemmas consider a sibling group $\{a_1, a_2, \dots, a_m\}$ of T_1 as in Theorem 1 and assume $F_2^{a_i}$ is not a component of F_2 , for all $1 \leq i \leq m$. Lemma 5 above covers the first of the five cases.

Lemma 7 (At Most One Pendant Subtree). *If a_1, a_2, \dots, a_r share a minimal LCA l in F_2 , $s_i = 1$, for $1 \leq i < r$, and a_r is a child of l , then there exist a binary resolution F of F_2 and an edge set E of size $e(T_1, T_2, F_2)$ such that $F \div E$ is an AF of T_1 and T_2 and either $\{e_{B_1}, e_{B_2}, \dots, e_{B_{r-1}}\} \subseteq E$ or $\{e_{B_1}, e_{B_2}, \dots, e_{B_{i-1}}, e_{B_{i+1}}, e_{B_{i+2}}, \dots, e_{B_{r-1}}\} \subseteq E$ and $a'_i \sim_{F \div E} b'_i$, for some $1 \leq i \leq r - 1$ and two leaves $a'_i \in F_2^{a_i}$ and $b'_i \in F_2^{B_i}$.*

Proof. Let F be a binary resolution of F_2 , and E an edge set of size $e(T_1, T_2, F_2)$ such that $F \div E$ is an AF of F_1 and F_2 , and assume F and E are chosen so that $|E \cap \{e_{a_1}, e_{a_2}, \dots, e_{a_r}, e_{B_1}, e_{B_2}, \dots, e_{B_{r-1}}\}|$ is maximized. Let $I := \{i \mid 1 \leq i \leq r - 1 \text{ and } E \cap \{e_{a_i}, e_{B_i}\} \neq \emptyset\}$ and $I' := \{i \mid 1 \leq i \leq r - 1 \text{ and } e_{a_i} \in E\}$.

First observe that $|I| \geq r - 2$. Otherwise there would exist two indices $1 \leq i < j \leq r - 1$ such that $E \cap \{e_{a_i}, e_{B_i}, e_{a_j}, e_{B_j}\} = \emptyset$. By the choice of E and Lemma 1, this would imply that there exist leaves $a'_i \in F_2^{a_i}$, $b'_i \in F_2^{B_i}$, $a'_j \in F_2^{a_j}$, and $b'_j \in F_2^{B_j}$ such that $a'_i \sim_{F \div E} b'_i$ and $a'_j \sim_{F \div E} b'_j$. If $a'_i \sim_{F \div E} a'_j$, then $a'_i b'_i | a'_j$ would be a triple of $F \div E$ incompatible with F_1 . If $a'_i \sim_{F \div E} b'_j$, the components of $F \div E$ containing a'_i and a'_j would overlap in F_1 . In both cases, $F \div E$ would not be an AF of F_1 and F_2 , a contradiction.

Since $|I| \geq r - 2$ and $i \notin I$ implies that there are two leaves $a'_i \in F_2^{a_i}$ and $b'_i \in F_2^{B_i}$ such that $a'_i \sim_{F \div E} b'_i$, the lemma follows if we can show that there exist a resolution F' of F_2 and an edge set E' of size $|E'| \leq |E|$ such that (i) $F' \div E'$ is an AF of F_1 and F_2 , (ii) $\{e_{B_i} \mid i \in I\} \subseteq E'$, and (iii) for any $1 \leq i \leq r - 1$, there exist leaves $a'_i \in F_2^{a_i}$ and $b'_i \in F_2^{B_i}$ such that $a'_i \sim_{F' \div E'} b'_i$ if and only if there exist leaves $a''_i \in F_2^{a_i}$ and $b''_i \in F_2^{B_i}$ such that $a''_i \sim_{F \div E} b''_i$.

Let Y be the set of leaves in all trees $F_2^{a_i}$, $i \in I' \cup \{r\}$, and $F_2^{B_i}$, $i \in I'$, let $Z := X \setminus Y \cup \{a'_r\}$, for an arbitrary leaf $a'_r \in F_2^{a_r}$, let l' be the LCA in F of all nodes in Y , and let E'' be the set of edges in E that belong to the paths between l' and the nodes in $\{a_i \mid i \in I' \cup \{r\}\}$. Since $e_{a_i} \in E$, for all $i \in I'$, we have $|E''| \geq |I'|$. We construct F' from F_2 by resolving every node set B_i where $i \in I'$, resolving the set $\{a_r\} \cup \{p_{a_i} \mid i \in I'\}$, and resolving all remaining multifurcations so that $F|Y = F'|Y$ and $F|Z = F'|Z$. We define the set E' to be $E' := E \setminus E'' \cup \{e_{B_i} \mid i \in I'\}$ if $|E''| = |I'|$; otherwise $E' := E \setminus E'' \cup \{e_{B_i} \mid i \in I'\} \cup \{e_{l''}\}$, where l'' is the LCA in F' of all nodes in Y . It is easily verified that F' and E' satisfy properties (ii) and (iii) and that $|E'| \leq |E|$. Thus, it remains to prove that $F' \div E'$ is an AF of F_1 and F_2 .

Any triple of $F' \div E'$ incompatible with F_1 has to involve exactly one leaf $a'_i \in F_2^{a_i}$, for some $i \in I'$, because any other triple exists either in $F \div E$ or in F_1 and, thus, is compatible with F_1 . Thus, any triple $a'_i | xy$ or $a'_i x | y$ of $F' \div E'$ incompatible with F_1 must satisfy $x, y \notin (F')^{l''}$ because $e_{B_i} \in E'$, for all $i \in I'$. If $|E''| > |I'|$, no such triple exists because $e_{l''} \in E'$. If $|E''| = |I'|$, observe

that $x, y \notin (F')^{l''}$ implies that $a'_r|xy$ or $a'_r|x|y$ is also a triple of $F' \div E'$ incompatible with F_1 . By the construction of F' , this triple is also a triple of F , and since $E \setminus E' = \{e_{a_i} \mid i \in I'\}$ and $x, y \notin F_2^{a_i}$, for all $i \in I'$, it is also a triple of $F \div E$ incompatible with F_1 , a contradiction.

If two components of $F' \div E'$ overlap in F_1 , let u, v, x, y be four leaves such that $u \sim_{F' \div E'} v \approx_{F' \div E'} x \sim_{F' \div E'} y$ and the two paths P_{uv} and P_{xy} between u and v and between x and y , respectively, share an edge. Let Y' be the set of leaves in the subtrees $F_2^{a_i}$, $i \in I' \cup \{r\}$, and assume P_{uv} has no fewer endpoints in Y' than P_{xy} .

If both endpoints of P_{uv} are in Y' , the two paths cannot overlap because a_1, a_2, \dots, a_r are siblings in F_1 and our choices of E and E' ensure that all leaves in the same subtree $F_2^{a_i}$ belong to the same component of $F' \div E'$.

If both P_{uv} and P_{xy} have one leaf in Y' , their corresponding paths in $F' \div E'$ include l'' . Thus, $u \sim_{F' \div E'} x$.

If neither P_{uv} nor P_{xy} has an endpoint in Y' , then $u \sim_{F \div E} v$ and $x \sim_{F \div E} y$. If P_{uv} has one endpoint in Y' , say $u \in Y'$, and P_{xy} does not have an endpoint in Y' , then $x \sim_{F \div E} y$ and, by the same arguments we used to show that no triple of $F' \div E'$ is incompatible with F_1 , there exists a leaf $a'_r \in F_2^{a_r}$ such that $a'_r \sim_{F \div E} v$ and the path from a'_r to v in F_1 overlaps P_{xy} . Thus, in both cases we have two paths $P_{u'v}$, $u' \in \{u, a'_r\}$, and P_{xy} in F_1 such that $u' \sim_{F \div E} v$ and $x \sim_{F \div E} y$ and the two paths overlap. Since no two components of $F \div E$ overlap in F_1 , these two paths belong to the same component of $F \div E$ and w.l.o.g. form the quartet $u'x|vy$, while $u' \sim_{F' \div E'} v \approx_{F' \div E'} x \sim_{F' \div E'} y$. Since $E' \setminus E \subseteq \{e_{B_i} \mid i \in I'\} \cup \{e_{l''}\}$, we either have $x, y \in F_2^{B_i}$ and $u', v \notin F_2^{a_i} \cup F_2^{B_i}$, for some $i \in I'$, or $u', v \in (F')^{l''}$, $u', v \notin F_2^{B_i}$, for all $i \in I'$, and $x, y \notin (F')^{l''}$. In the former case, these four leaves form the quartet $u'v|xy$ in $F \div E$, a contradiction. In the latter case, we have $u', v \in Y'$, and we already argued that this case is impossible. \square

Lemma 8 (One Pendant Subtree). *If a_1, a_2, \dots, a_r share a minimal LCA l in F_2 , $m > 2$, and $s_i = 1$, for all $1 \leq i \leq r$, then there exist a resolution F of F_2 and an edge set E of size $e(T_1, T_2, F_2)$ such that $F \div E$ is an AF of T_1 and T_2 and either $\{e_{a_1}, e_{a_2}, \dots, e_{a_r}\} \subseteq E$, $\{e_{B_1}, e_{B_2}, \dots, e_{B_r}\} \subseteq E$ or there exists an index $1 \leq i \leq r$ and two leaves $a'_i \in F_2^{a_i}$ and $b'_i \in F_2^{B_i}$ such that $a'_i \sim_{F \div E} b'_i$ and either $\{e_{a_1}, e_{a_2}, \dots, e_{a_{i-1}}, e_{a_{i+1}}, e_{a_{i+2}}, \dots, e_{a_r}\} \subseteq E$ or $\{e_{B_1}, e_{B_2}, \dots, e_{B_{i-1}}, e_{B_{i+1}}, e_{B_{i+2}}, \dots, e_{B_r}\} \subseteq E$.*

Proof. Let F be a resolution of F_2 , and E an edge set of size $e(T_1, T_2, F_2)$ such that $F \div E$ is an AF of F_1 and F_2 , and assume F and E are chosen so that $|E \cap \{e_{a_1}, e_{a_2}, \dots, e_{a_r}, e_{B_1}, e_{B_2}, \dots, e_{B_r}\}|$ is maximized.

If there exists an index $1 \leq j \leq r$ such that $e_{B_j} \in E$, then assume w.l.o.g. that $j = r$. The forests F_1 and $F'_2 := F_2 \div \{e_{B_r}\}$ satisfy the conditions of Lemma 7. Hence, there exist a resolution F' of F'_2 and an edge set E' of size $e(T_1, T_2, F'_2) = e(T_1, T_2, F_2) - 1$ such that $F' \div E'$ is an AF of F_1 and F'_2 and, thus, of F_1 and F_2 and either $\{e_{B_1}, e_{B_2}, \dots, e_{B_{r-1}}\} \subseteq E'$ or $a'_i \sim_{F' \div E'} b'_i$ and $\{e_{B_1}, e_{B_2}, \dots, e_{B_{i-1}}, e_{B_{i+1}}, e_{B_{i+2}}, \dots, e_{B_{r-1}}\} \subseteq E'$, for some index $1 \leq i \leq r - 1$ and leaves $a'_i \in F_2^{a_i}$ and $b'_i \in F_2^{B_i}$. Thus, the resolution F'' of F_2 such that $F'' \div \{e_{B_r}\} = F'$ and the edge set $E' \cup \{e_{B_r}\}$ satisfy the lemma.

If $E \cap \{e_{B_1}, e_{B_2}, \dots, e_{B_r}\} = \emptyset$, then by the same arguments as in Lemma 7, we can have at most one index $1 \leq i \leq r$ such that $a'_i \sim_{F \div E} b'_i$, for two leaves $a'_i \in F_2^{a_i}$ and $b'_i \in F_2^{B_i}$. If such an index i exists, then $\{e_{a_1}, e_{a_2}, \dots, e_{a_{i-1}}, e_{a_{i+1}}, e_{a_{i+2}}, \dots, e_{a_r}\} \subseteq E$. If no such index exists, then $\{e_{a_1}, e_{a_2}, \dots, e_{a_r}\} \subseteq E$. \square

Lemma 9 (Multiple Pendant Subtrees, $m = 2$). *If $\{a_1, a_2\}$ is a sibling group such that a_1 and a_2 share a minimal LCA l in F_2 and $s_1 + s_2 \geq 2$, then there exist a resolution F of F_2 and an edge set E of size $e(T_1, T_2, F_2)$ such that $F \div E$ is an AF of T_1 and T_2 and either $E \cap \{e_{a_1}, e_{a_2}\} \neq \emptyset$ or $\{e_{B_{11}}, e_{B_{12}}, \dots, e_{B_{1s_1}}\} \cup \{e_{B_{21}}, e_{B_{22}}, \dots, e_{B_{2s_2}}\} \subseteq E$.*

Proof. We prove this by induction on $s = s_1 + s_2$. The base case is $s = 1^3$ and the claim follows from Lemma 6.

Having established the base case, we can assume $s > 1$ and the lemma holds for all $1 \leq s' < s$. By Theorem 1, there exist a resolution F of F_2 and an edge set E of size $e(T_1, T_2, F_2)$ such that $F \div E$ is an AF of F_1 and F_2 and $E \cap \{e_{a_1}, e_{a_2}, e_{B_1}, e_{B_2}\} \neq \emptyset$. Assume F and E are chosen so that $|E \cap \{e_{a_1}, e_{a_2}, e_{B_1}, e_{B_2}\}|$ is maximized. If $E \cap \{e_{a_1}, e_{a_2}\} \neq \emptyset$, the lemma holds, so assume the contrary. If $s_2 = 0$ and $E \cap \{e_{a_1}, e_{a_2}, e_{B_1}, e_{B_2}\} = \{e_{B_2}\}$, the choice of F and E and Lemma 1 imply that there exist leaves $a'_1 \in F_2^{a_1}$, $b'_1 \in F_2^{B_1}$, $a'_2 \in F_2^{a_2}$, and $x \notin F_2^{a_2}$ such that $a'_1 \sim_{F_2 \div \{e_{B_2}\}} b'_1$ and $a'_2 \sim_{F_2 \div \{e_{B_2}\}} x$. Thus, a_1 and a_2 satisfy the conditions of Lemma 5 after cutting edge e_{B_2} , which implies that we can choose F and E so that $E \cap \{e_{a_1}, e_{a_2}\} \neq \emptyset$ in addition to $e_{B_2} \in E$, a contradiction. Now assume $s_2 \neq 0$ or $E \cap \{e_{a_1}, e_{a_2}, e_{B_1}, e_{B_2}\} \neq \{e_{B_2}\}$. In this case, $E \cap \{e_{B_{11}}, e_{B_{21}}\} \neq \emptyset$. Assume w.l.o.g. that $e_{B_{11}} \in E$. Then the inductive hypothesis shows that there exist a resolution F' of F_2 and an edge set E' of size $e(T_1, T_2, F_2)$ such that $F' \div E'$ is an AF of F_1 and $F_2 \div \{e_{B_{11}}\}$ (and, hence, of F_1 and F_2) such that either $E' \cap \{e_{a_1}, e_{a_2}\} \neq \emptyset$ or $\{e_{B_{11}}, e_{B_{12}}, \dots, e_{B_{1s_1}}\} \cup \{e_{B_{21}}, e_{B_{22}}, \dots, e_{B_{2s_2}}\} \subseteq E'$. Thus, the lemma also holds in this case. \square

The proofs of the following two lemmas are similar to that of Lemma 9. They are provided in Sections S6.3 and S6.4 of the supplementary material.

Lemma 10 (Multiple Pendant Subtrees, $m > 2$ and $r > 2$). *If a_1, a_2, \dots, a_r share a minimal LCA l in F_2 , $m > 2$, $r > 2$, and $s_1 \geq 2$, then there exist a resolution F of F_2 and an edge set E of size $e(T_1, T_2, F_2)$ such that $F \div E$ is an AF of T_1 and T_2 and $E \cap \{e_{a_1}, e_{a_2}\} \neq \emptyset$, $\{e_{B_{11}}, e_{B_{12}}, \dots, e_{B_{1s_1}}\} \subseteq E$ or $\{e_{B_{21}}, e_{B_{22}}, \dots, e_{B_{2s_2}}\} \subseteq E$.*

Lemma 11 (Multiple Pendant Subtrees, $m > 2$ and $r = 2$). *If a_1, a_2, \dots, a_r share a minimal LCA l in F_2 , $m > 2$, $r = 2$ and $s_1 \geq 2$, then there exist a resolution F of F_2 and an edge set E of size $e(T_1, T_2, F_2)$ such that $F \div E$ is an AF of T_1 and T_2 and $E \cap \{e_{a_1}, e_{a_2}\} \neq \emptyset$, $\{e_{B_{11}}, e_{B_{12}}, \dots, e_{B_{1s_1}}\} \subseteq E$ or $\{e_{B_{21}}, e_{B_{22}}, \dots, e_{B_{2s_2}}, e_{B'_2}\} \subseteq E$, where B'_2 is the set of siblings of a_i after cutting $e_{B_{21}}, e_{B_{22}}, \dots, e_{B_{2s_2}}$.*

4 MAF Algorithms

In this section, we present an FPT algorithm for computing MAFs of multifurcating rooted trees. This algorithm also forms the basis for a 3-approximation algorithm with running time $O(n \log n)$, which is presented in Section S8 of the supplementary material.

As is customary for FPT algorithms, we focus on the decision version of the problem: “Given two rooted X -trees T_1 and T_2 and a parameter k , is $d_{sPR}(T_1, T_2) \leq k$?” To compute the distance between two trees, we start with $k = 0$ and increase it until we receive an affirmative answer. This does not increase the running time of the algorithm by more than a constant factor, as the running time depends exponentially on k .

Our FPT algorithm is recursive. Each invocation $\text{MAF}(F_1, F_2, k, a_0)$ takes two (partially resolved) forests F_1 and F_2 of T_1 and T_2 , a parameter k , and (optionally) a node a_0 that exists in F_1 and F_2 as inputs. F_1 is the union of a tree \hat{T}_1 and a forest F_0 disjoint from \hat{T}_1 , while F_2 is the union of the same forest F_0 and another forest \hat{F}_2 with the same label set as \hat{T}_1 . The output of the invocation $\text{MAF}(F_1, F_2, k, a_0)$ satisfies two conditions: (i) If $e(T_1, T_2, F_2) > k$, the output is “no”. (ii) If $e(T_1, T_2, F_2) \leq k$ and either $a_0 = \text{nil}$ or there exists an MAF F of F_1 and F_2 such that a_0

³We excluded this case from the statement of the lemma, in order to keep the cases covered by the different lemmas disjoint, but the lemma also holds for $s = 1$. A similar comment applies to Lemma 10.

is not a root of F and $a_0 \approx_F a_i$, for every sibling a_i of a_0 in F_1 , the output is “yes”. Since the top-level invocation is $\text{MAF}(T_1, T_2, k, \text{nil})$, these two conditions ensure that this invocation decides whether $e(T_1, T_2, T_2) \leq k$.

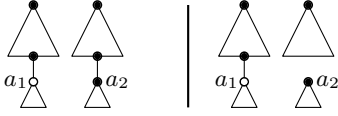
The representation of the input to each recursive call includes two sets of labelled nodes: R_d (roots-done) contains the roots of F_0 , R_t (roots-todo) contains the roots of (not necessarily maximal) subtrees that agree between \dot{T}_1 and \dot{F}_2 . We refer to the nodes in these sets by their labels. For the top-level invocation, $F_1 = \dot{T}_1 = T_1$, $F_2 = \dot{F}_2 = T_2$, and $F_0 = \emptyset$; R_d is empty and R_t contains all leaves of T_1 ; $a_0 = \text{nil}$.

$\text{MAF}(F_1, F_2, k, a_0)$ uses the results from Section 3 to identify a small collection $\{E_1, E_2, \dots, E_q\}$ of subsets of edges of \dot{F}_2 such that $e(T_1, T_2, F_2) \leq k$ only if $e(T_1, T_2, F_2 \div E_i) \leq k - |E_i|$, for at least one $1 \leq i \leq q$. It calls $\text{MAF}(F_1, F_2 \div E_i, k - |E_i|, a'_i)$ recursively, for each subset E_i and an appropriate parameter a'_i , and returns “yes” if and only if one of these recursive calls does.

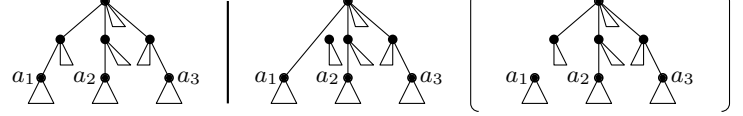
A naïve use of the structural results from Section 3 would explore many overlapping edge subsets. For example, one branch of the algorithm may cut an edge e_{a_i} and then an edge e_{a_j} , while a sibling branch may cut e_{a_j} and then e_{a_i} . As we hinted at in Section 3, if we cut edge e_{a_i} or its sibling edge e_{B_i} in two sibling invocations, then there is no need to consider cutting either of these two edges in their sibling invocations or their descendants. Using the results of Lemmas 7–11, we obtain more generally: if we cut e_{a_i} or its set of progressive siblings $\{e_{B_{i1}}, e_{B_{i2}}, \dots, e_{B_{is_i}}\}$ in two sibling invocations, then we need not consider these edge sets in their sibling invocations or their descendants. Thus, we set $a_0 = a_i$ in these sibling invocations and thereby instruct the algorithm to ignore these edges as candidates for cutting. An invocation $\text{MAF}(F_1, F_2, k, a_0)$ with $a_0 \neq \text{nil}$ (Step 7 below) makes only two recursive calls when it would make significantly more recursive calls if $a_0 = \text{nil}$ (Step 8 below). This is not a trivial change, as it is required to obtain the running time claimed in Theorem 2. The steps of our procedure are as follows.

1. (Failure) If $k < 0$, then $e(T_1, T_2, F_2) \geq 0 > k$. Return “no” in this case.
2. (Success) If $|R_t| = 1$, then $F_1 = F_2$. Hence, F_2 is an AF of T_1 and T_2 , that is, $e(T_1, T_2, F_2) = 0 \leq k$. Return “yes” in this case.
3. (Prune maximal agreeing subtrees) If there is no node $r \in R_t$ that is a root in F_2 , proceed to Step 4. Otherwise choose such a node $r \in R_t$; remove it from R_t and add it to R_d , thereby moving the corresponding subtree of \dot{F}_2 to F_0 ; and cut the edge e_r in F_1 . If r ’s parent p_r in F_1 now has only one child, contract p_r . If $a_0 \neq \text{nil}$ and p_r ’s only child before the contraction was a_0 , set $a_0 = \text{nil}$. Note that these changes affect only F_1 . Thus, $e(T_1, T_2, F_2)$ remains unchanged. Return to Step 2.
4. Choose a sibling group $\{a_1, a_2, \dots, a_m\}$ in \dot{T}_1 such that $a_1, a_2, \dots, a_m \in R_t$. If two or more members of the sibling group chosen in this invocation’s parent invocation remain in \dot{T}_1 , choose that sibling group.
5. (Grow agreeing subtrees) While there exist indices $1 \leq i < j \leq m$ such that a_i and a_j are siblings in \dot{F}_2 , do the following: Remove a_i and a_j from R_t ; resolve a_i and a_j in \dot{T}_1 and \dot{F}_2 ; label their new parent in both forests with (a_i, a_j) and add it to R_t . The new node (a_i, a_j) becomes a member of the current sibling group and m decreases by 1. If $m = 1$ after resolving all such sibling pairs $\{a_i, a_j\}$, contract the parent of the only remaining member of the sibling group and return to Step 2; otherwise proceed to Step 6.
6. If $a_i \approx_{F_2} a_j$, for all $1 \leq i < j \leq m$, proceed to Step 7. Otherwise there exists a node l that is a minimal LCA of a group of nodes in the current sibling group. If the most recent minimal LCA

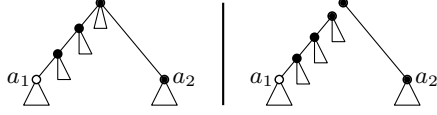
Case 7.1



Case 7.2



Case 7.3



Case 7.4

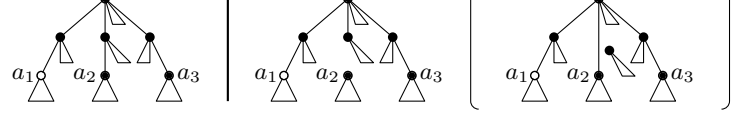


Figure 5: The different cases of Step 7. Only the subtree of \dot{F}_2 rooted in l is shown. The left side shows a possible input for each case, the right side visualizes the cuts made in each recursive call. The node a_0 is shown as a hollow circle (if it is a descendant of l).

chosen in an ancestor invocation is a minimal LCA of a subset of nodes in the current sibling group, choose l to be this node; otherwise choose l arbitrarily. Now order the nodes in the sibling group $\{a_1, a_2, \dots, a_m\}$ so that, for some $r \geq 2$, a_1, a_2, \dots, a_r are descendants of l , while, for all $1 \leq i \leq r < j \leq m$, either the LCA of a_i and a_j is a proper ancestor of l or $a_i \approx_{F_2} a_j$. Order a_1, a_2, \dots, a_r so that $s_1 \geq s_2 \geq \dots \geq s_r$. (Recall that s_i is the number of nodes on the path from a_i to l , excluding a_i and l .) The order of $a_{r+1}, a_{r+2}, \dots, a_m$ is arbitrary.

7. (Two-way branching) If $a_0 = \text{nil}$, proceed to Step 8. Otherwise distinguish four cases, where $x = 1$ if $a_1 \neq a_0$, and $x = 2$ otherwise (see Figure 5).

7.1. If $a_1 \approx_{F_2} a_i$, for all $i \neq 1$, and $a_2 \approx_{F_2} a_j$, for all $j \neq 2$, call $\text{MAF}(F_1, F_2 \div \{e_{a_x}\}, k-1, a_0)$.

7.2. If a_1, a_2, \dots, a_r share the minimal LCA l in F_2 and a_0 is not a descendant of l , call $\text{MAF}(F_1, F_2 \div \{e_{B_{11}}, e_{B_{12}}, \dots, e_{B_{1s_1}}\}, k-s_1, a_0)$. If $s_1 > 1$ or $s_r > 0$, also call $\text{MAF}(F_1, F_2 \div \{e_{a_1}\}, k-1, a_0)$.

7.3. If $r = 2$, $s_x = 0$, a_0 is a descendant of l , and either l is a root of F_2 or its parent has a member a_i of the current sibling group as a child, call $\text{MAF}(F_1, F_2 - \{e_{B_x}\}, k-1, a_0)$.

7.4. If a_1, a_2, \dots, a_r share the minimal LCA l in F_2 , a_0 is a descendant of l , and Case 7.3 does not apply, call $\text{MAF}(F_1, F_2 \div \{e_{a_x}\}, k-1, a_0)$. If $m > 2$ and $r > 2$, make another recursive call $\text{MAF}(F_1, F_2 \div \{e_{B_{x1}}, e_{B_{x2}}, \dots, e_{B_{xs_x}}\}, k-s_x, a_0)$. If $m > 2$ but $r = 2$, the second recursive call is $\text{MAF}(F_1, F_2 \div \{e_{B_{x1}}, e_{B_{x2}}, \dots, e_{B_{xs_x}}, e_{B'_x}\}, k-(s_x+1), a_0)$, where B'_x is the set of siblings of x after cutting $e_{B_{x1}}, e_{B_{x2}}, \dots, e_{B_{xs_x}}$.

Return “yes” if one of the recursive calls does; otherwise return “no”.

8. (Unconstrained branching) Distinguish seven cases and choose the first case that applies (see Figure 6):

8.1. If $a_1 \approx_{F_2} a_i$, for all $i \neq 1$, and $a_2 \approx_{F_2} a_j$, for all $j \neq 2$, call $\text{MAF}(F_1, F_2 \div \{e_{a_1}\}, k-1, \text{nil})$ and $\text{MAF}(F_1, F_2 \div \{e_{a_2}\}, k-1, \text{nil})$.

8.2. If $s_i = 1$, for $1 \leq i < r$, and a_r is a child of l , call $\text{MAF}(F_1, F_2 \div \{e_{B_1}, e_{B_2}, \dots, e_{B_{r-1}}\}, k-(r-1), \text{nil})$ and $\text{MAF}(F_1, F_2 \div \{e_{B_1}, e_{B_2}, \dots, e_{B_{i-1}}, e_{B_{i+1}}, e_{B_{i+2}}, \dots, e_{B_{r-1}}\}, k-(r-2), a_i)$, for all $1 \leq i \leq r-1$.

- 8.3. If $m = 2$, and $s_1 + s_2 \geq 2$, call $\text{MAF}(F_1, F_2 \div \{e_{a_1}\}, k - 1, \text{nil})$, $\text{MAF}(F_1, F_2 \div \{e_{a_2}\}, k - 1, \text{nil})$, and $\text{MAF}(F_1, F_2 \div \{e_{B_{11}}, e_{B_{12}}, \dots, e_{B_{1s_1}}\} \cup \{e_{B_{21}}, e_{B_{22}}, \dots, e_{B_{2s_2}}\}, k - (s_1 + s_2), \text{nil})$.
- 8.4. If $m > 2$, $r = 2$, and $s_1 = s_2 = 1$, call $\text{MAF}(F_1, F_2 \div \{e_{a_1}, e_{a_2}\}, k - 2, \text{nil})$, $\text{MAF}(F_1, F_2 \div \{e_{B_1}, e_{B_2}\}, k - 2, \text{nil})$, $\text{MAF}(F_1, F_2 \div \{e_{B_1}, e_{B'_1}\}, k - 2, a_2)$, and $\text{MAF}(F_1, F_2 \div \{e_{B_2}, e_{B'_2}\}, k - 2, a_1)$, where B'_i is the set of siblings of a_i after cutting edge B_i . If l is a root, l 's parent p_l has at least one child that is neither l nor a member a_j of the current sibling group or l 's grandparent has at least one child that is neither p_l nor a member a_h of the current sibling group, make two additional calls $\text{MAF}(F_1, F_2 \div \{e_{a_1}\}, k - 1, a_2)$ and $\text{MAF}(F_1, F_2 \div \{e_{a_2}\}, k - 1, a_1)$.
- 8.5. If $m > 2$, $r > 2$, and $s_i = 1$, for all $1 \leq i \leq r$, call $\text{MAF}(F_1, F_2 \div \{e_{a_1}, e_{a_2}, \dots, e_{a_r}\}, k - r, \text{nil})$, $\text{MAF}(F_1, F_2 \div \{e_{B_1}, e_{B_2}, \dots, e_{B_r}\}, k - r, \text{nil})$, $\text{MAF}(F_1, F_2 \div \{e_{a_1}, e_{a_2}, \dots, e_{a_{i-1}}, e_{a_{i+1}}, e_{a_{i+2}}, \dots, e_{a_r}\}, k - (r - 1), a_i)$, for all $1 \leq i \leq r$, and $\text{MAF}(F_1, F_2 \div \{e_{B_1}, e_{B_2}, \dots, e_{B_{i-1}}, e_{B_{i+1}}, e_{B_{i+2}}, \dots, e_{B_r}\}, k - (r - 1), a_i)$, for all $1 \leq i \leq r$.
- 8.6. If $m > 2$, $r = 2$, $s_1 \geq 2$, $s_2 = 0$ and either l is a root of F_2 or its parent has a member a_i of the current sibling group as a child, call $\text{MAF}(F_1, F_2 - \{e_{a_1}\}, k - 1, \text{nil})$, $\text{MAF}(F_1, F_2 - \{e_{B_{11}}, e_{B_{12}}, \dots, e_{B_{1s_1}}\}, k - s_1, \text{nil})$, and $\text{MAF}(F_1, F_2 - \{e_{B_2}\}, k - 1, \text{nil})$.
- 8.7. If $m > 2$, $s_1 \geq 2$, and Case 8.6 does not apply, call $\text{MAF}(F_1, F_2 \div \{e_{a_1}\}, k - 1, \text{nil})$, $\text{MAF}(F_1, F_2 \div \{e_{a_2}\}, k - 1, a_1)$, and $\text{MAF}(F_1, F_2 \div \{e_{B_{11}}, e_{B_{12}}, \dots, e_{B_{1s_1}}\}, k - s_1, \text{nil})$. If $r > 2$, call $\text{MAF}(F_1, F_2 \div \{e_{B_{21}}, e_{B_{22}}, \dots, e_{B_{2s_2}}\}, k - s_2, a_1)$. If $r = 2$, call $\text{MAF}(F_1, F_2 \div \{e_{B_{21}}, e_{B_{22}}, \dots, e_{B_{2s_2}}, e_{B'_2}\}, k - (s_2 + 1), a_1)$, where B'_2 is defined as in Lemma 11.

Return “yes” if one of the recursive calls does; otherwise return “no”.

Theorem 2. *Given two rooted X -trees T_1 and T_2 and a parameter k , it takes $O((1 + \sqrt{2})^k n) = O(2.42^k n)$ time to decide whether $e(T_1, T_2, T_2) \leq k$.*

Proof. We use the algorithm in this section, invoking it as $\text{MAF}(T_1, T_2, k, \text{nil})$. We leave its correctness proof to a separate lemma (Lemma 12 below) and focus on bounding its running time here. As we argue in Section S7 of the supplementary material, each invocation $\text{MAF}(F_1, F_2, k', a_0)$ takes $O(n)$ time. Thus, it suffices to bound the number of invocations by $O((1 + \sqrt{2})^k)$. Let $I(k, t)$ be the number of invocations that are descendants of an invocation $\text{MAF}(F_1, F_2, k, a_0)$ in the recursion tree, where $t = 1$ if the invocation executes Step 7 but not Step 8; otherwise $t = 0$. We develop a recurrence relation for $I(k, t)$ and use it to show that $I(k, t) \leq (1 + \sqrt{2})^{2 + \max(0, k - t + 3)} + 2(t - 1)$, which proves our claim.⁴

An invocation with $t = 0$ by definition either executes neither Step 7 nor Step 8, or it executes Step 8. By considering the different cases of Step 8, we obtain the following recurrence for the case

⁴This is a fairly loose bound on $I(k, t)$, but it is easy to manipulate.

when $t = 0$:

$$I(k, 0) \leq \begin{cases} 1 & \text{no recursion} \\ 1 + 2I(k-1, 0) & \text{Case 8.1} \\ 1 + I(k-1, 0) + I(k, 1) & \text{Case 8.2} \\ 1 + 2I(k-1, 0) + I(k-2, 0) & \text{Cases 8.3, 8.6} \\ 1 + 2I(k-2, 0) + 2I(k-1, 1) \\ \quad + 2I(k-2, 1) & \text{Case 8.4} \\ 1 + 2I(k-3, 0) + 3I(k-2, 0) \\ \quad + 3I(k-2, 1) & \text{Case 8.5} \\ 1 + I(k-1, 0) + I(k-2, 0) \\ \quad + 2I(k-1, 1) & \text{Case 8.7} \end{cases}$$

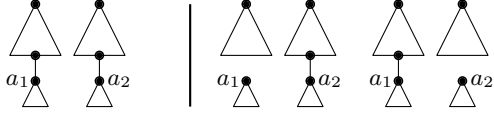
The values of the first arguments of all $I(\cdot, \cdot)$ terms are easily verified for Cases 8.1, 8.3, 8.4, and 8.6. For Case 8.2, we observe that $r \geq 2$ and the worst case arises when $r = 2$, giving the claimed recurrence. For Case 8.7, we observe again that $r \geq 2$. If $r > 2$, then $s_2 > 0$, giving the recurrence for this case. If $r = 2$, then s_2 may be 0, but the fourth recursive call cuts $s_2 + 1$ edges, thereby giving the same recurrence as when $r > 2$. For Case 8.5, finally, we have $r \geq 3$ and, once again, the minimum value, $r = 3$, is the worst case, which gives the recurrence.

Next we argue about the correctness of all second arguments that are 1 in these recurrences. Each such term $I(\cdot, 1)$ corresponds to a recursive call with $a_0 \neq \text{nil}$. Thus, in order to justify setting $t = 1$, we need to show that each such invocation executes Step 7 but not Step 8, which follows if in this child invocation, the current invocation's sibling group exists and at least one additional edge cut in F_2 is required to make this sibling group agree between F_1 and F_2 —we say that the sibling group agrees between F_1 and F_2 if F_2 does not contain a triple incompatible with F_1 and involving descendants of at least two members of the sibling group, and there are no two paths between leaves in F_2 that (i) belong to different components of F_2 , (ii) overlap in F_1 , and (iii) have at least one endpoint each that is a descendant of a member of the current sibling group. The only cases that make recursive calls with $a_0 \neq \text{nil}$ are Cases 8.2, 8.4, 8.5 and 8.7. We consider each case in turn.

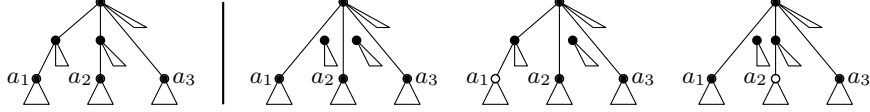
In Case 8.2, consider a recursive call that cuts edges $e_{B_1}, e_{B_2}, \dots, e_{B_{i-1}}, e_{B_{i+1}}, e_{B_{i+2}}, \dots, e_{B_{r-1}}$, for some $1 \leq i \leq r-1$. Assume w.l.o.g. that $i = 1$. After cutting edges $e_{B_2}, e_{B_3}, \dots, e_{B_{r-1}}$, there still exist leaves $a'_1 \in F_2^{a_1}$, $b'_1 \in F_2^{B_1}$, and $a'_r \in F_2^{a_r}$ such that F_1 contains the triple $a'_1 a'_r | b'_1$ while F_2 contains the triple $a'_1 b'_1 | a'_r$. Thus, the sibling group does not agree between F_1 and F_2 yet.

In Case 8.4, if we make only four recursive calls, we can ignore whether setting $a_0 = a_1$ or $a_0 = a_2$ in the third and fourth recursive calls translates into $t = 1$ for these recursive calls because even the recurrence $I(k, 0) = 1 + 4I(k-2, 0)$ is bounded by the recurrence $I(k, 0) = 1 + 2I(k-2, 0) + 2I(k-2, 1) + 2I(k-1, 1)$ for the case when we make six recursive calls. If we make six recursive calls, there exists a member a_3 of the current sibling group that is not a descendant of l . Thus, the conditions for making the fifth and sixth recursive calls imply that there exist leaves $a'_3 \in F_2^{a_3}$ and $b'_3 \notin F_2^{a_i}$, for all $1 \leq i \leq m$, such that $a'_3 \sim_{F_2} b'_3$ and the path from a'_3 to b'_3 in F_2 is disjoint from F_2^l . After cutting e_{B_1} and $e_{B'_1}$, there exist leaves $a'_2 \in F_2^{a_2}$ and $b'_2 \in F_2^{B_2}$ such that $a'_2 \sim_{F_2 \setminus \{e_{B_1}, e_{B'_1}\}} b'_2$. Thus, we have two paths in different connected components (between a'_2 and b'_2 and between a'_3 and b'_3) that overlap in F_1 , and the sibling group does not agree between F_1 and F_2 yet. After cutting e_{a_1} , we obtain overlapping paths as above if $a_2 \sim_{F_2} a_3$; otherwise $a'_2 b'_2 | a'_3$ is a triple of F_2 incompatible with F_1 . Thus, once again the sibling group does not agree between F_1 and F_2 . Similar arguments show that setting $t = 1$ is correct when cutting edge e_{a_2} or edges e_{B_2} and $e_{B'_2}$.

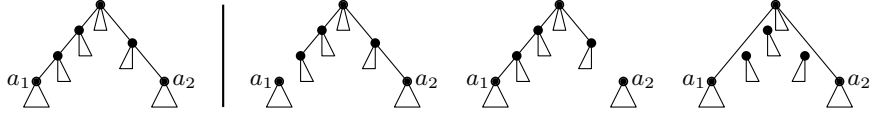
Case 8.1



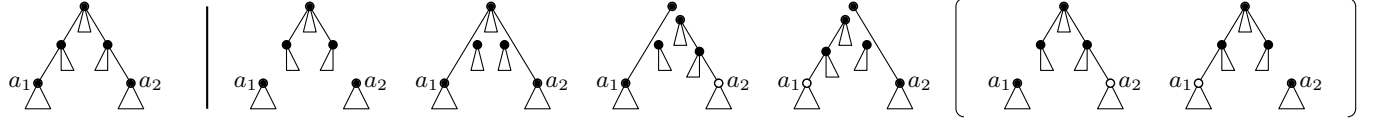
Case 8.2



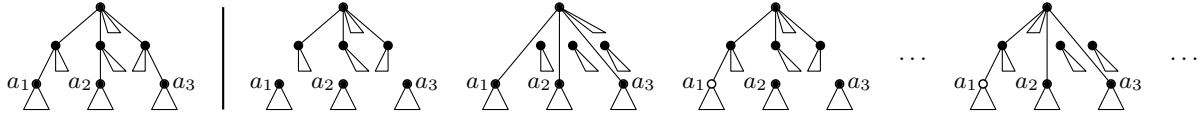
Case 8.3



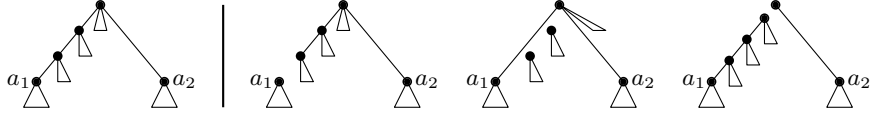
Case 8.4



Case 8.5



Case 8.6



Case 8.7

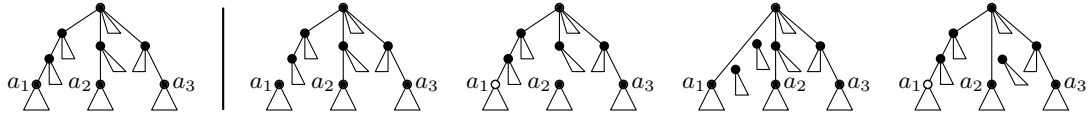


Figure 6: The different cases of Step 8. Only the subtree of \dot{F}_2 rooted in l is shown. The left side shows a possible input for each case, the right side visualizes the cuts made in each recursive call. Whenever $a_0 \neq \text{nil}$ in a recursive call, it is shown as a hollow circle. The last two calls in Case 8.4 may or may not be made.

In Case 8.5, we claim that it is correct to set $t = 1$ for each recursive call that cuts edges $e_{B_1}, e_{B_2}, \dots, e_{B_{i-1}}, e_{B_{i+1}}, e_{B_{i+2}}, \dots, e_{B_r}$, for some $1 \leq i \leq r$. Assume w.l.o.g. that $i = 1$. After cutting edges $e_{B_2}, e_{B_3}, \dots, e_{B_r}$, there exist leaves $a'_1 \in F_2^{a_1}$, $b'_1 \in F_2^{B_1}$, and $a'_2 \in F_2^{a_2}$ such that $a'_1 a'_2 | b'_1$ is a triple of F_1 and $a'_1 b'_1 | a'_2$ is a triple of F_2 . Thus, the sibling group does not agree between F_1 and F_2 yet.

In Case 8.7, observe that l is not a root and its parent does not have a member of the current sibling group as a child. Thus, since $m > 2$, there exists an index j and two leaves $a'_j \in F_2^{a_j}$ and $b'_j \notin F_2^{a_h}$, for all $1 \leq h \leq m$, such that $a'_j \sim_{F_2} b'_j$ and the path from a'_j to b'_j in F_2 is disjoint from the path between any two leaves in $F_2^{a_1}$ and $F_2^{B_{11}}$. After cutting e_{a_2} , there exist leaves $a'_1 \in F_2^{a_1}$ and $b'_1 \in F_2^{B_{11}}$ such that $a'_1 \sim_{F_2 \div \{e_{a_2}\}} b'_1$. If $a'_1 \sim_{F_2} a'_j$, we thus have a triple $a'_1 b'_1 | a'_j$ of F_2 incompatible with F_1 . If $a'_1 \not\sim_{F_2} a'_j$, the path between a'_1 and b'_1 overlaps the path between a'_j and b'_j in F_1 . In either case, the current sibling group does not agree between F_1 and F_2 yet. This concludes the correctness proof of the recurrence for $I(k, 0)$.

For $t = 1$, we distinguish whether or not the current invocation \mathcal{I} makes a recursive call with $t = 0$ and whether it makes one or two recursive calls. If \mathcal{I} makes no recursive call with $t = 0$, we obtain $I(k, 1) \leq 1 + 2I(k - 1, 1)$ because each case of Step 7 makes at most two recursive calls, with parameters no greater than $k - 1$. If \mathcal{I} makes only one recursive call, with $t = 0$, we obtain $I(k, 1) \leq 1 + I(k - 1, 0)$ because this recursive call has parameter no greater than $k - 1$. Finally, if \mathcal{I} makes two recursive calls, at least one of them with $t = 0$, \mathcal{I} must have applied Case 7.2 or 7.4. Let \mathcal{I}' be one of the invocations \mathcal{I} makes with $t = 0$. If $t = 0$ for invocation \mathcal{I}' because \mathcal{I}' terminates in Step 1 or 2, we obtain $I(k, 1) \leq 2 + I(k - 1, 0)$ by counting invocations \mathcal{I} and \mathcal{I}' and the number of recursive calls spawned by the sibling invocation of \mathcal{I}' , which cannot be more than $I(k - 1, 0)$. So assume that $t = 0$ for invocation \mathcal{I}' and that \mathcal{I}' does make further recursive calls. Then the sibling group chosen in invocation \mathcal{I} must agree between the input forests of invocation \mathcal{I}' .

If invocation \mathcal{I} applies Case 7.2 and makes two recursive calls, we observe that $m \geq 3$ because a_0 is a member of \mathcal{I} 's sibling group, a_0 is not a descendant of l , and l has at least two descendants in the sibling group. Furthermore, $s_1 > 0$. Thus, after cutting e_{a_1} , a_2 has a sibling forest B'_2 that does not include a_0 . Since a_0 also has a sibling forest B_0 that does not include a_2 , \mathcal{I} 's sibling group cannot agree between \hat{T}_1 and \hat{F}_2 after cutting e_{a_1} . This implies that $t = 1$ for the first recursive call $\text{MAF}(F_1, F_2 \div \{e_{a_1}\}, k - 1, a_0)$, and \mathcal{I}' is the second recursive call $\text{MAF}(F_1, F_2 \div \{e_{B_{11}}, e_{B_{12}}, \dots, e_{B_{1s_1}}\}, k - s_1, a_0)$. This gives the recurrence $I(k, 1) = 1 + I(k - 1, 1) + I(k - s_1, 0)$. Since no two members of \mathcal{I} 's sibling group are siblings in F_2 and a_0 is not a descendant of l , cutting edges $e_{B_{11}}, e_{B_{12}}, \dots, e_{B_{1s_1}}$ can make \mathcal{I} 's sibling group agree between F_1 and F_2 only if $r = 2$, $s_2 = 0$, and either l is a root of F_2 or the only pendant nodes of the path from l to the root of its component in F_2 are members of \mathcal{I} 's sibling group. Thus, since we assume we make two recursive calls, we must have $s_1 \geq 2$, that is, the recurrence for this case is $I(k, 1) \leq 1 + I(k - 1, 1) + I(k - 2, 0)$.

Finally, if invocation \mathcal{I} applies Case 7.4, observe that, since a_0 is a descendant of l and has a group of sibling trees B_0 that do not contain any member a_i of \mathcal{I} 's sibling group, this sibling group can be made to agree between F_1 and F_2 only by cutting e_{a_x} . Moreover, since no member a_i of \mathcal{I} 's sibling group is a root of F_2 , cutting e_{a_x} can make this sibling group agree between F_1 and F_2 only if $m = 2$. Thus, Case 7.4 makes only one recursive call and we obtain $I(k, 1) = 1 + I(k - 1, 0)$ in this case.

By combining the different possibilities for the case when $t = 1$, we obtain the recurrence

$$I(k, 1) \leq \max(1 + 2I(k - 1, 1), 2 + I(k - 1, 0), 1 + I(k - 1, 1) + I(k - 2, 0)).$$

Simple substitution now shows that $I(k, t) \leq (1 + \sqrt{2})^{2 + \max(0, k - t + 3)} + 2(t - 1)$. \square

Lemma 12. *For two rooted X -trees T_1 and T_2 and a parameter k , the invocation $\text{MAF}(T_1, T_2, k, \text{nil})$ returns “yes” if and only if $d_{\text{SPR}}(T_1, T_2) = e(T_1, T_2, T_2) \leq k$.*

Proof. We use induction on k to prove the following two claims, which together imply the lemma:

(i) If $e(T_1, T_2, F_2) > k$, the invocation $\text{MAF}(F_1, F_2, k, a_0)$ returns “no”. (ii) If $e(T_1, T_2, F_2) \leq k$ and either $a_0 = \text{nil}$ or there exists an MAF F of F_1 and F_2 such that a_0 is not a root of F and $a_0 \approx_F a_i$, for every sibling a_i of a_0 in F_1 , the invocation $\text{MAF}(F_1, F_2, k, a_0)$ returns “yes”.

(i) Assume $e(T_1, T_2, F_2) > k$. If $k < 0$, the invocation returns “no” in Step 1. If $k \geq 0$, assume for the sake of contradiction that the invocation returns “yes”. If it does so in Step 2, then F_2 is an AF of T_1 and T_2 , that is, $e(T_1, T_2, F_2) = 0 \leq k$, a contradiction. Otherwise it returns “yes” in Step 7 or 8. Thus, there exists a child invocation $\text{MAF}(F'_1, F'_2, k', a'_0)$ that returns “yes”, where $F'_2 = F_2 \div E$ and $k' = k - |E|$, for some non-empty edge set E . By the inductive hypothesis, we therefore have $e(T_1, T_2, F'_2) \leq k'$ and, hence, $e(T_1, T_2, F_2) \leq k' + |E| = k$, again a contradiction.

(ii) Assume $e(T_1, T_2, F_2) \leq k$ and either $a_0 = \text{nil}$ or there exists an MAF F of F_1 and F_2 such that a_0 is not a root of F and $a_0 \approx_F a_i$, for every sibling a_i of a_0 in F_1 . In particular, $k \geq 0$ and the invocation $\text{MAF}(F_1, F_2, k, a_0)$ produces its answer in Step 2, 7 or 8. If it produces its answer in Step 2, it answers “yes”. Next we consider Steps 7 and 8 and prove that at least one of the recursive calls made in each case returns “yes”, which implies that the current invocation returns “yes”.

In Step 7, $a_0 \neq \text{nil}$, that is, a_0 is not a root of F and $a_0 \approx_F a_i$, for every sibling a_i of a_0 in F_1 . In Case 7.1, $a_x \approx_{F_2} a_i$ and, hence, $a_x \approx_F a_i$, for all $i \neq x$. Thus, a_x is a root of F because otherwise the components of F containing a_x and a_0 would overlap in F_1 . This implies that there exists an edge set E such that $e_{a_x} \in E$ and $F = F_2 \div E$, that is, the recursive call $\text{MAF}(F_1, F_2 \div \{e_{a_x}\}, k - 1, a_0)$ returns “yes”.

In Case 7.2, $a'_1 \approx_F b'_1$, for all leaves $a'_1 \in F_2^{a_1}$ and $b'_1 \in F_2^{B_{1j}}$, $1 \leq j \leq s_1$, because the path between a'_1 and b'_1 would overlap the component containing a_0 . Thus, there exists an edge set E such that $F_2 \div E = F$ and either $e_{a_1} \in E$ or $\{e_{B_{11}}, e_{B_{12}}, \dots, e_{B_{1s_1}}\} \subseteq E$. If we make two recursive calls in Case 7.2, this shows that one of them returns “yes”. If we make only one recursive call, we have $s_1 = 1$ and $s_r = 0$. Assume this recursive call returns “no”. Then $e_{a_1} \in E$. Let F' be the forest obtained by cutting e_{B_1} instead of e_{a_1} , resolving the pair $\{a_1, a_r\}$, cutting edge $e_{(a_1, a_r)}$ instead of e_{a_r} if $e_{a_r} \in E$, and otherwise cutting the same edges as in E . F and F' are identical, except that in F' , a_1 and a_r are siblings and no leaf in $F_2^{B_1}$ can reach a leaf not in $F_2^{B_1}$. The former cannot introduce any triples incompatible with F_1 because a_1 and a_r are siblings in F_1 . The latter cannot introduce any overlapping components because this would imply that $a'_1 \sim_F b'_1$, for two leaves $a'_1 \in F_2^{a_1}$ and $b'_1 \in F_2^{B_1}$, and we already argued that no such path can exist. Thus, F' is also an MAF of F_1 and F_2 . Finally, a_0 is not a root in F' and $a_0 \approx_{F'} a_1$ because otherwise $a_0 \sim_F a_r$. Thus, since there exists an edge set E' such that $F' = F_2 \div E'$ and $e_{B_1} \in E'$, the recursive call $\text{MAF}(F_1, F_2 \div \{e_{B_1}\}, k - 1, a_0)$ returns “yes”, a contradiction.

In Case 7.3, there exists an edge set E such that $F = F_2 \div E$ and $E \cap \{e_{a_x}, e_{B_x}\} \neq \emptyset$. Otherwise we would have $a_x \sim_F a_0$ or $a'_x \sim_F b'_x$, for two leaves $a'_x \in F_2^{a_x}$ and $b'_x \notin F_2^{a_i}$, for all $1 \leq i \leq m$, but we have $a_x \approx_F a_0$ and the path between a'_x and b'_x would overlap the component of F that contains a_0 . Now, if $e_{B_x} \notin E$, we construct an MAF F' of F_1 and F_2 such that a_0 is not a root in F' and $a_0 \approx_{F'} a_i$, for all $1 \leq i \leq m$, and an edge set E' such that $F' = F_2 \div E'$ and $e_{B_x} \in E'$ as in Case 7.2. Thus, the invocation $\text{MAF}(F_1, F_2 \div \{e_{B_x}\}, k - 1, a_0)$ returns “yes”.

In Case 7.4, finally, one of the two recursive calls $\text{MAF}(F_1, F_2 \div \{e_{a_x}\}, k - 1, a_0)$ or $\text{MAF}(F_1, F_2 \div \{e_{B_{x1}}, e_{B_{x2}}, \dots, e_{B_{xs_x}}\}, k - s_x, a_0)$ must return “yes”, by the same arguments as in Case 7.2. Thus, if $m > 2$ and $r > 2$, one of the recursive calls we make returns “yes”. If $m > 2$ and $r = 2$, we observe that after cutting edges $e_{B_{x1}}, e_{B_{x2}}, \dots, e_{B_{xs_x}}$, a_x and a_0 satisfy the conditions of Case 7.3, which shows that we can cut edge B'_x immediately after cutting these edges. Thus, if the recursive call $\text{MAF}(F_1, F_2 \div$

$\{e_{a_x}\}, k-1, a_0)$ returns “no”, the recursive call $\text{MAF}(F_1, F_2 \div \{e_{B_{x1}}, e_{B_{x2}}, \dots, e_{B_{xsx}}, e_{B'_x}\}, k-s_x, a_0)$ must return “yes” in this case. Finally, if $m=2$, since $a_x \sim_F a_0$ and any path between two leaves $a'_x \in F_2^{a_x}$ and $b'_x \notin F_2^{a_x} \cup F_2^{a_0}$ would overlap the component of F that contains a_0 , a_x is a root in F . Thus, the invocation $\text{MAF}(F_1, F_2 \div \{e_{a_x}\}, k-1, a_0)$ returns “yes” in this case.

Now consider Step 8. In Cases 8.1, 8.2, 8.3, and 8.5, Lemmas 5, 7, 9, 8 and the inductive hypothesis show that one of the recursive calls returns “yes” and, thus, the current invocation returns “yes”.

In Case 8.4, Lemma 8 and the inductive hypothesis show that one of the recursive calls $\text{MAF}(F_1, F_2 \div \{e_{a_1}, e_{a_2}\}, k-2, \text{nil})$, $\text{MAF}(F_1, F_2 \div \{e_{B_1}, e_{B_2}\}, k-2, \text{nil})$, $\text{MAF}(F_1, F_2 \div \{e_{B_1}\}, k-1, a_2)$, $\text{MAF}(F_1, F_2 \div \{e_{B_2}\}, k-1, a_1)$, $\text{MAF}(F_1, F_2 \div \{e_{a_1}\}, k-1, a_2)$ or $\text{MAF}(F_1, F_2 \div \{e_{a_2}\}, k-1, a_1)$ would return “yes”. Thus, we need to argue only that we can cut *both* e_{B_i} and $e_{B'_i}$ in the third and fourth recursive calls, and that the last two recursive calls are not necessary when we do not make them.

First consider cutting e_{B_1} and setting $a_0 = a_2$ in the third recursive call. We require this call to return “yes” only if the other calls return “no”. The forest $F'_2 = F_2 \div \{e_{B_1}\}$ contains a triple $a'_1|a'_2b'_2$, where $a'_1 \in F_2^{a_1}$, $a'_2 \in F_2^{a_2}$, and $b'_2 \in F_2^{B_2}$, while F_1 contains the triple $a'_1a'_2|b'_2$. Thus, in order to obtain an MAF of F_1 and F'_2 where a_2 exists and is not a root, we need to cut either e_{a_1} or $e_{B'_1}$. Since the first and fifth recursive calls return “no”, however, we know that cutting a_1 cannot lead to an MAF of F_1 and F_2 , and we can cut $e_{B'_1}$ along with edge e_{B_1} . The case when we cut $e_{B'_2}$ along with edge e_{B_2} is analogous.

If we do not make the recursive call $\text{MAF}(F_1, F_2 \div \{e_{a_1}\}, k-1, a_2)$, then l has exactly one sibling, a_i , and its parent p_l is either a root or has exactly one sibling, a_j . Thus, the forest $F'_2 = F_2 \div \{e_{a_1}\}$ contains a triple $a'_2b'_2|a'_i$, where $a'_2 \in F_2^{a_2}$, $b'_2 \in F_2^{B_2}$, and $a'_i \in F_2^{a_i}$, while F_1 contains the triple $a'_2a'_i|b'_2$. In order to obtain an MAF of F_1 and F'_2 where a_2 exists and is not a root, we therefore need to cut either e_{a_i} or $e_{B_i} = e_l$. If p_l is a root, cutting either edge has the same effect. If p_l has a sibling a_j and we cut e_{a_i} , we can obtain an alternate MAF by cutting e_l instead of e_{a_i} , resolving $\{a_i, a_j\}$, cutting edge $e_{(a_i, a_j)}$ instead of e_{a_j} if $e_{a_j} \in E$, and otherwise cutting the same edges as in E . Thus, we can always replace the recursive call $\text{MAF}(F_1, F_2 \div \{e_{a_1}\}, k-1, a_2)$ with the call $\text{MAF}(F_1, F_2 \div \{e_{a_1}, e_l\}, k-2, a_2)$ without affecting the correctness of the algorithm. If this call returns “yes”, however, then so does the call $\text{MAF}(F_1, F_2 \div \{e_{B_1}, e_{B'_1}\}, k-2, a_2)$ because we can yet again obtain an alternate MAF by cutting edges e_{B_1} and e_{B_2} instead of edges e_{a_1} and e_l , resolving $\{a_1, a_i\}$, cutting $e_{(a_1, a_i)}$ instead of e_{a_i} if $e_{a_i} \in E$, and otherwise cutting the same edges as in E . Thus, the call $\text{MAF}(F_1, F_2 \div \{e_{a_1}\}, k-1, a_2)$ can be eliminated altogether. An analogous argument shows that we can eliminate the call $\text{MAF}(F_1, F_2 \div \{e_{a_2}\}, k-1, a_1)$.

In Case 8.6, Lemma 11 and the inductive hypothesis show that one of the recursive calls $\text{MAF}(F_1, F_2 \div \{e_{a_1}\}, k-1, \text{nil})$, $\text{MAF}(F_1, F_2 \div \{e_{a_2}\}, k-1, \text{nil})$, $\text{MAF}(F_1, F_2 \div \{e_{B_{11}}, e_{B_{12}}, \dots, e_{B_{1s_1}}\}, k-s_1, \text{nil})$ or $\text{MAF}(F_1, F_2 \div \{e_{B_2}\}, k-1, \text{nil})$ would return “yes”. We need to show that the call $\text{MAF}(F_1, F_2 \div \{e_{a_2}\}, k-1, \text{nil})$ is not necessary. To see this, observe that, if l is a root, then cutting e_{a_2} or e_{B_2} has the same effect. If l is not a root but has a member a_i of the current sibling group as a sibling, then we can obtain an alternate MAF by cutting e_{B_2} instead of e_{a_2} , resolving $\{a_2, a_i\}$, cutting $e_{(a_2, a_i)}$ instead of e_{a_i} if $a_i \in E$, and otherwise cutting the same edges as in E .

In Case 8.7, finally, the correctness follows from Lemmas 10 and 11 if we can show that setting $a_0 \neq a_1$ is correct for the second and fourth recursive calls. This, however, follows because, if neither the first nor the third recursive call returns “yes”, then in every MAF F of F_1 and F_2 , a_1 exists and there exist two leaves $a'_1 \in F_2^{a_1}$ and $b'_1 \in F_2^{B_{1j}}$, for some $1 \leq j \leq s_1$, such that $a'_1 \sim_F b'_1$. \square

The proof of the following theorem is provided in Section S8 of the supplementary material.

Theorem 3. *Given two rooted X -trees T_1 and T_2 , a 3-approximation of $e(T_1, T_2, T_2) = d_{SPR}(T_1, T_2)$ can be computed in $O(n \log n)$ time.*

5 Conclusions

We developed efficient algorithms for computing MAFs of multifurcating trees. Our fixed-parameter algorithm achieves the same running time as in the binary case and our 3-approximation algorithm achieves a running time of $O(n \log n)$, almost matching the linear running time for the binary case. Implementing and testing our algorithms will be the focus of future work.

Two other directions to be explored by future work are practical improvements of the running time of the FPT algorithm presented here and extending our FPT algorithm so it can be used to compute maximum *acyclic* agreement forests (MAAFs) and, hence, the hybridization number of multifurcating trees. To speed up our FPT algorithm for computing MAFs, it may be possible to extend the reduction rules used by Linz and Semple [16] for computing MAAF of multifurcating trees so they can be applied to MAF computations, and combine them with the FPT algorithm in this paper. The fastest fixed-parameter algorithms for computing MAAF of binary trees [10–12] are extensions of the binary MAF algorithms of Whidden et al. [8, 9]. These algorithms were developed by examining which search branches of the binary MAF algorithm get “stuck” with cyclic agreement forests and consider cutting additional edges to avoid these cycles [10] or refine cyclic agreement forests to acyclic agreement forests [11, 12]. Similarly, Chen and Wang [21] recently extended the MAF fixed-parameter algorithm for two binary trees to compute agreement forests of multiple binary trees using an iterative branching approach. The proofs of various structural lemmas in this paper prove that *any* MAF can be obtained by cutting an edge set that includes certain edges. To prove this, we started with an arbitrary MAF and an edge set such that cutting these edges yields this MAF, and then we modified this edge set so that it includes the desired set of edges without changing the resulting forest. As in the binary case [12], the same lemmas apply also to MAAFs; since the modifications in the proofs do not change the resulting AF, the only needed change in the proof is to start with an edge set such that cutting it yields an MAAF. Thus, numerous lemmas in this paper may also form the basis for an efficient algorithm for computing MAAFs.

Finally, we note that our fixed-parameter algorithm becomes greatly simplified when comparing a binary tree to a multifurcating tree. This is common in practice when, for example, comparing many multifurcating gene trees to a binary reference tree or binary supertree. To see this, suppose F_1 is binary, that is, $m = 2$ in every case of the FPT algorithm. Then only Cases 8.1, 8.2 and 8.3 apply in Step 8 and Step 7 never applies. Using our observation that cutting e_{B_2} is not necessary in Case 8.2 when $m = 2$, our algorithm becomes similar to the MAF algorithm for binary trees [9]. We further note, in the interest of practical efficiency, that cutting a_2 is unnecessary in this case when the parent of a_1 is a binary node (and, indeed, our algorithm is then identical to the algorithm of [9] when applied to two binary trees).

Acknowledgement

Chris Whidden was supported by a Killam predoctoral scholarship and the Tula Foundation, Robert G. Beiko is a Canada Research Chair and was supported by NSERC, Genome Atlantic, and the Canada Foundation for Innovation, and Norbert Zeh is a Canada Research Chair and was supported by NSERC and the Canada Foundation for Innovation.

References

- [1] D. M. Hillis, C. Moritz, and B. K. Mable, Eds., *Molecular Systematics*. Sinauer Associates, 1996.
- [2] V. Rosas-Magallanes, P. Deschavanne, L. Quintana-Murci, R. Brosch, B. Gicquel, and O. Neyrolles, “Horizontal transfer of a virulence operon to the ancestor of mycobacterium tuberculosis,” *Molecular Biology and Evolution*, vol. 23, no. 6, pp. 1129–1135, 2006.
- [3] M. Baroni, S. Grünwald, V. Moulton, and C. Semple, “Bounding the number of hybridisation events for a consistent evolutionary history,” *Journal of Mathematical Biology*, vol. 51, no. 2, pp. 171–182, 2005.
- [4] R. G. Beiko and N. Hamilton, “Phylogenetic identification of lateral genetic transfer events,” *BMC Evolutionary Biology*, vol. 6, no. 1, p. 15, 2006.
- [5] M. Bordewich and C. Semple, “On the computational complexity of the rooted subtree prune and regraft distance,” *Annals of Combinatorics*, vol. 8, no. 4, pp. 409–423, 2005.
- [6] G. Hickey, F. Dehne, A. Rau-Chaplin, and C. Blouin, “SPR distance computation for unrooted trees,” *Evolutionary Bioinformatics*, vol. 4, pp. 17–27, 2008.
- [7] M. Bordewich and C. Semple, “Computing the minimum number of hybridization events for a consistent evolutionary history,” *Discrete Applied Mathematics*, vol. 155, no. 8, pp. 914–928, 2007.
- [8] C. Whidden and N. Zeh, “A unifying view on approximation and FPT of agreement forests,” in *Proceedings of the 9th International Workshop, WABI 2009*, ser. Lecture Notes in Bioinformatics, vol. 5724. Springer-Verlag, 2009, pp. 390–401.
- [9] C. Whidden, R. G. Beiko, and N. Zeh, “Fast FPT algorithms for computing rooted agreement forests: Theory and experiments,” in *Proceedings of the 9th International Symposium on Experimental Algorithms, SEA 2010*, ser. Lecture Notes in Computer Science, vol. 6049. Springer-Verlag, 2010, pp. 141–153.
- [10] B. Albrecht, C. Scornavacca, A. Cenci, and D. H. Huson, “Fast computation of minimum hybridization networks,” *Bioinformatics*, vol. 28, no. 2, pp. 191–197, 2012.
- [11] Z. Chen and L. Wang, “Hybridnet: a tool for constructing hybridization networks,” *Bioinformatics*, vol. 26, no. 22, pp. 2912–2913, 2010.
- [12] C. Whidden, R. G. Beiko, and N. Zeh, “Fixed-Parameter and Approximation Algorithms for Maximum Agreement Forests,” *ArXiv e-prints*, Aug. 2011.
- [13] E. M. Rodrigues, M.-F. Sagot, and Y. Wakabayashi, “The maximum agreement forest problem: Approximation algorithms and computational experiments,” *Theoretical Computer Science*, vol. 374, no. 1-3, pp. 91–110, 2007.
- [14] M. Bordewich, C. McCartin, and C. Semple, “A 3-approximation algorithm for the subtree distance between phylogenies,” *Journal of Discrete Algorithms*, vol. 6, no. 3, pp. 458–471, 2008.
- [15] J. Hein, T. Jiang, L. Wang, and K. Zhang, “On the complexity of comparing evolutionary trees,” *Discrete Applied Mathematics*, vol. 71, no. 1-3, pp. 153–169, 1996.

- [16] S. Linz and C. Semple, “Hybridization in nonbinary trees,” *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, vol. 6, pp. 30–45, 2009.
- [17] L. van Iersel, S. Kelk, N. Lekić, and L. Stougie, “Computing nonbinary agreement forests,” *ArXiv e-prints*, Oct. 2012.
- [18] W. Maddison, “Reconstructing character evolution on polytomous cladograms,” *Cladistics*, vol. 5, no. 4, pp. 365–377, 1989.
- [19] B. L. Allen and M. Steel, “Subtree transfer operations and their induced metrics on evolutionary trees,” *Annals of Combinatorics*, vol. 5, no. 1, pp. 1–15, 2001.
- [20] M. L. Bonet, K. St. John, R. Mahindru, and N. Amenta, “Approximating subtree distances between phylogenies,” *Journal of Computational Biology*, vol. 13, no. 8, pp. 1419–1434, 2006.
- [21] Z.-Z. Chen and L. Wang, “Algorithms for reticulate networks of multiple phylogenetic trees,” *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, vol. 9, pp. 372–384, 2012.
- [22] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to Algorithms, Second Edition*. McGraw-Hill, Jul. 2001.

S6 Omitted Proofs

S6.1 Lemma 3

The only difference between F_2 and F'_2 is the expansion of $\{a_{p+1}, a_{p+2}, \dots, a_m\}$ in F'_2 , so $e(F_1, F_2, F_2) \leq e(F_1, F_2, F'_2)$. Since $F \div E$ is an MAF of F_1 and F_2 , it suffices to show that $F \div E$ is an AF of F_1 and F'_2 to prove that $e(F_1, F_2, F'_2) \leq e(F_1, F_2, F_2)$. Assume the contrary. Then, since $F \div E$ is a forest of F_1 , it cannot be a forest of F'_2 . Since the only difference between F_2 and F'_2 is the expansion of $\{a_{p+1}, a_{p+2}, \dots, a_m\}$, this implies that some component of $F \div E$ contains leaves $a'_i \in F_2^{a_i}$ and $a'_j \in F_2^{a_j}$, for some $1 \leq i \leq p$ and $p+1 \leq j \leq m$, contradicting that $a'_i \sim_{F \div E} a'_j$ for all such leaves.

S6.2 Lemma 6

As in the proof of Theorem 1, (ii) follows immediately from (i), so it suffices to prove that there exist a binary resolution F of F_2 and an edge set E of size $e(T_1, T_2, F_2)$ such that $F \div E$ is an MAF of T_1 and T_2 (and, hence, of F_1 and F_2) and $E \cap \{e_{a_1}, e_{a_2}, e_{B_1}\} \neq \emptyset$. Once again, we show that, if $E \cap \{e_{a_1}, e_{a_2}, e_{B_1}\} = \emptyset$, we can replace an edge $f \in E$ with an edge in $\{e_{a_1}, e_{a_2}, e_{B_1}\}$ without changing $F \div E$.

This follows from the same arguments as in the proof of Theorem 1 unless there exist leaves $a'_1 \in F_2^{a_1}$, $a'_2 \in F_2^{a_2}$ and $b'_1 \in F_2^{B_1}$ such that $a'_1 \sim_{F \div E} p_{a_1} \sim_{F \div E} b'_1$ and $a'_2 \sim_{F \div E} p_{a_2}$. In this case, since a_1 is not a child of l , we have $a_2 \notin F_2^{B_1}$. Thus, since $\{a_1, a_2\}$ is a sibling pair in F_1 , F_1 contains the triple $a'_1 a'_2 | b'_1$. Since F_2 contains the triple $a'_1 b'_1 | a'_2$ and $a'_1 \sim_{F \div E} b'_1$, this implies that $a'_1 \sim_{F \div E} a'_2$. Thus, we also have $a'_2 \sim_{F \div E} x$, for all $x \in F_2^l \setminus F_2^{a_2}$, as otherwise the components of $F \div E$ containing a'_1, b'_1 and a'_2, x would overlap in F_1 . We choose an arbitrary leaf $b'_2 \in B_2$ and the first edge $f \in E$ on the path from p_{a_2} to b'_2 . Lemma 1 implies that $F \div E = F \div (E \setminus \{f\} \cup \{e_{a_2}\})$.

S6.3 Lemma 10

As in the proof of Lemma 9, we prove this using induction on $s = s_1 + s_2$. The base case is $s = 2$ and, hence, $s_1 = s_2 = 1$ because $r > 2$ implies that $s_1 > 0$ and $s_2 > 0$. In this case, Theorem 1 proves the lemma. So assume $s > 2$ and the claim holds for all $1 \leq s' < s$. By Theorem 1, there exist a resolution F of F_2 and an edge set E of size $e(T_1, T_2, F_2)$ such that $F \div E$ is an AF of F_1 and F_2 and $E \cap \{e_{a_1}, e_{a_2}, e_{B_{11}}, e_{B_{21}}\} \neq \emptyset$. Using an inductive argument as in the proof of Lemma 9, it follows that there exist a resolution F' of F_2 and an edge set E' satisfying the lemma.

S6.4 Lemma 11

First consider the case when $s_2 = 0$, which is possible because $r = 2$. Then $B'_2 = B_2$ and, by Theorem 1, there exist a resolution F of F_2 and an edge set E of size $e(T_1, T_2, F_2)$ such that $F \div E$ is an AF of T_1 and T_2 and $E \cap \{e_{a_1}, e_{a_2}, e_{B_{11}}, e_{B_2}\} \neq \emptyset$. If $E \cap \{e_{a_1}, e_{a_2}, e_{B_2}\} \neq \emptyset$, the lemma holds. Otherwise $e_{B_{11}} \in E$ and an inductive argument similar to the one in the proof of Lemma 9 proves the lemma.

If $s_2 > 0$, we observe that the proof of Lemma 10 did not use the assumption that $r > 2$ but only that it implies $s_2 > 0$. Hence, this proof shows that there exist a resolution F of F_2 and an edge set E of size $e(T_1, T_2, F_2)$ such that $F \div E$ is an AF of T_1 and T_2 and $E \cap \{e_{a_1}, e_{a_2}\} \neq \emptyset$, $\{e_{B_{11}}, e_{B_{12}}, \dots, e_{B_{1s_1}}\} \subseteq E$ or $\{e_{B_{21}}, e_{B_{22}}, \dots, e_{B_{2s_2}}\} \subseteq E$. Among all such resolutions and edge sets, choose F and E so that $E \cap \{e_{a_1}, e_{a_2}\} \neq \emptyset$ or $\{e_{B_{11}}, e_{B_{12}}, \dots, e_{B_{1s_1}}\} \subseteq E$ if possible. If we can find such a pair (F, E) , the lemma holds. Otherwise $\{e_{B_{21}}, e_{B_{22}}, \dots, e_{B_{2s_2}}\} \subseteq E$. Let F' be the forest obtained from F_2 by resolving $B_{21}, B_{22}, \dots, B_{2s_2}$ and cutting edges $e_{B_{21}}, e_{B_{22}}, \dots, e_{B_{2s_2}}$. In F' , we have $r = 2$ and $s_2 = 0$. Hence, by the argument in the previous paragraph, there exist a resolution F'' of F' and an edge set E'' of size $e(T_1, T_2, F')$ such that $F'' \div E''$ is an AF of T_1 and T_2 and $E'' \cap \{e_{a_1}, e_{a_2}\} \neq \emptyset$, $\{e_{B_{11}}, e_{B_{12}}, \dots, e_{B_{1s_1}}\} \subseteq E''$ or $e_{B'_2} \in E''$. In the first two cases, we obtain a contradiction to the choice of F and E . In the latter case, the set $\{e_{B_{21}}, e_{B_{22}}, \dots, e_{B_{2s_2}}\} \cup E''$ has size $s_2 + e(T_1, T_2, F'') = e(T_1, T_2, F_2)$, $F_2 \div (\{e_{B_{21}}, e_{B_{22}}, \dots, e_{B_{2s_2}}\} \cup E'')$ is an AF of T_1 and T_2 , and $\{e_{B_{21}}, e_{B_{22}}, \dots, e_{B_{2s_2}}, e_{B'_2}\} \subseteq \{e_{B_{21}}, e_{B_{22}}, \dots, e_{B_{2s_2}}\} \cup E''$. Thus, the lemma holds in this case as well.

S7 Linear Time Per Invocation

We represent each forest as a collection of nodes, each of which points to its parent, to its leftmost child, and to its left and right siblings. This allows us to cut an edge in constant time, given the parent and child connected by this edge. Every labelled node (i.e., every node in R_t or R_d) stores a pointer to its counterpart in the other forest. For \dot{T}_1 , we maintain a list of sibling groups of labelled nodes. For each such group, the list stores a pointer to the parent of the sibling group, which allows us to access the members of the sibling group by traversing the list of the parent's children. To detect the creation of such a sibling group, and add it to the list, each internal node of \dot{T}_1 stores the number of its unlabelled children. When labelling a non-root node, we decrease its parent's unlabelled children count by one. If this count is now 0, the children of this parent node form a new sibling group, and we add a pointer to the parent to the list of sibling groups. For \dot{F}_2 , we maintain a list $R'_d \subseteq R_t$ of labelled nodes that are roots of \dot{F}_2 . This list is used to move these roots from R_t to R_d .

Steps 1–4 are implemented similarly to the algorithm for binary trees [9]. Step 1 clearly takes constant time. In Step 2, we can test in constant time whether $|R_t| \leq 1$ by inspecting at most two nodes in the first two sibling groups. Step 3 takes constant time to test whether the root list R'_d

is empty and, if it is not, cut the appropriate edge in \dot{T}_1 and update a constant number of lists and pointers. Step 4 takes constant time using the list of sibling groups. We always choose the next sibling group from the beginning of this list and append new sibling groups to the end. This automatically gives preference to the most recently chosen sibling group as required in Step 4.

Step 5 requires some care to implement efficiently. We iterate over the members a_1, a_2, \dots, a_m of the current sibling group and mark their parents in \dot{F}_2 . Initially, all nodes in \dot{F}_2 are unmarked. When inspecting a node a_i whose parent p_{a_i} in \dot{F}_2 is unmarked, we mark p_{a_i} with a_i . If p_{a_i} is already marked with a node a_j ($j < i$), then a_i and a_j are siblings in \dot{T}_1 and \dot{F}_2 . We resolve them in constant time and mark p_{a_i} (which is now the grandparent of a_i) with the new parent (a_i, a_j) of a_i and a_j . Since we spend constant time per member a_i of the sibling group, this procedure takes $O(m)$ time. Once it finishes, the remaining members of the sibling group are not siblings in \dot{F}_2 . Performing a contraction if the remaining sibling group has only one member takes constant time.

In Step 6, we perform a linear-time traversal of \dot{F}_2 to label every node x with the number r_x of members of the current sibling group among its descendants. Then, if the previously chosen minimal LCA l still exists in \dot{F}_2 and has at least two descendants in the current sibling group, we keep this choice of l . Otherwise a node x is a minimal LCA of a subset of the current sibling group if and only if $r_x \geq 2$ and $r_y \leq 1$, for each child y of x . If there is no such node x , we proceed to Step 7 without choosing l because $a_i \approx_{F_2} a_j$, for all $1 \leq i < j \leq m$. Otherwise we pick any node x satisfying this condition as the new minimal LCA l . No matter whether l is the previously chosen minimal LCA or a new node, we set $r = r_l$ and traverse the paths from l to its descendant members of the sibling group, a_1, a_2, \dots, a_r . We do this by visiting all descendants y of l such that $r_y = 1$. For all $1 \leq i \leq r$, the length of the path from l to a_i , excluding l and a_i , is s_i . We sort a_1, a_2, \dots, a_r by their path lengths s_1, s_2, \dots, s_r using Counting Sort [22]. Since $s_i \leq n$, for all $1 \leq i \leq r$, this takes linear time.

To distinguish between Steps 7 and 8, it suffices to examine a_0 . We distinguish between the cases in Steps 7 and 8 using the values of r , m , and s_1, s_2, \dots, s_r and, in Step 7, by testing whether a_0 is among the descendants of l . In each case, we can easily copy the forests, cut the appropriate edges, and update our lists and pointers in linear time for each of the recursive calls.

To summarize: Each execution of Steps 1–4 takes constant time. Step 1 is executed once per invocation. Steps 2–4 are executed at most a linear number of times per invocation because each execution, except the first one, is the result of finding a root of \dot{F}_2 in Step 3 or resolving sibling pairs in Step 5, both of which can happen only $O(n)$ times. Each execution of Step 5 takes $O(m)$ time. In a given invocation, Step 5 is executed at most once per sibling group (because we either proceed to Step 6 or return to Step 2 after completely resolving the sibling group). Thus, since the total size of all sibling groups is bounded by $|\dot{T}_1|$, the total cost of all executions of Step 5 per invocation is $O(n)$. Steps 6–8 are executed at most once per invocation and take linear time. Thus, each invocation of the algorithm takes linear time.

S8 A 3-Approximation Algorithm for Rooted MAF

We now show how to modify the FPT algorithm from Section 4 to obtain a 3-approximation algorithm with running time $O(n \log n)$. This algorithm is easy to implement iteratively, and this may be preferable in practice. In order to minimize the differences to the FPT algorithm, however, we describe it as a recursive algorithm. There are four differences to the FPT algorithm:

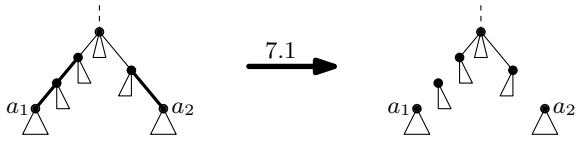
- Instead of deciding whether $e(T_1, T_2, F_2) \leq k$, an invocation $\text{MAF}(F_1, F_2)$ returns an integer k'' such that $e(T_1, T_2, F_2) \leq k'' \leq 3e(T_1, T_2, F_2)$. Thus, there is no need for a parameter k to

the invocation or for an equivalent of Step 1 of the FPT algorithm, and whenever Step 2 of the FPT algorithm would have returned “yes”, we now return 0 as our approximation k'' of $e(T_1, T_2, F_2)$ because F_2 is an AF of T_1 and T_2 .

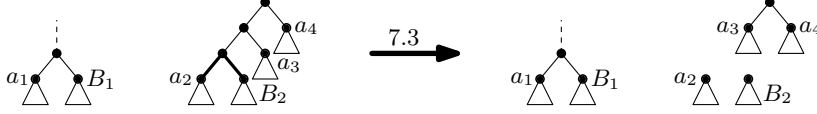
- We execute Step 5 only if the immediately preceding execution of Step 4 chose a new sibling group. This ensures that this step is executed only once per sibling group. As discussed in Section S7, the cost per sibling group $\{a_1, a_2, \dots, a_m\}$ is $O(m)$. Thus, the total cost of all executions of Step 5 in all recursive invocations is $O(n)$. After executing Step 5, implemented as discussed in Section S7, every node p in F_2 has at most one node a_i as a child, which is stored as p 's *representative* r_p . This allows us to merge sibling pairs $\{a_i, a_j\}$ that arise as a result of edge cuts after we executed Step 5 without re-executing this step: Whenever we contract a degree-2 vertex whose only child is a node a_i in the current sibling group and whose parent is p , we set $r_p := a_i$ if $r_p = \text{nil}$; otherwise we resolve the sibling pair $\{a_i, r_p\}$ as in Step 5 and store (a_i, r_p) as p 's new representative.
- We do not execute Step 6, as this would require linear time per invocation. Instead, we assign a *depth estimate* d_x to each node x and use it to choose the order in which to inspect the members of the current sibling group. Initially, d_x is x 's depth in T_2 , which is easily computed in linear time for all nodes $x \in T_2$. In general, d_x is one more than the depth of p_x in T_2 , where p_x is x 's parent in F_2 . In particular, d_x is an upper bound on x 's depth in F_2 and, for two nodes x and y with LCA l , we have $d_y > d_x$ if x is a child of l and y is not. When choosing a new sibling group in Step 4, we insert all group members into a max-priority queue Q , with their depth estimates as their priorities. When contracting the degree-2 parent p_x of a node x , we set $d_x := d_{p_x}$. If x is a member of the current sibling group, we update its priority in Q . When resolving a sibling pair $\{a_i, a_j\}$, we remove a_i and a_j from Q , set $d_{(a_i, a_j)} := d_{a_i}$, and insert (a_i, a_j) into Q . Finally, when cutting an edge e_{a_i} , for a member a_i of the current sibling group, we remove a_i from Q . These updates take $O(\log n)$ time per modification of F_2 . Since we modify F_2 at most $O(n)$ times, the total cost of all priority queue operations is $O(n \log n)$.
- We do not distinguish between Steps 7 and 8 (having no concept of a_0) and also do not distinguish between the various cases of the steps. Instead, we have a single Step 7 with four cases that each make one recursive call (see Figure S7).
 - 7.1. If $m = 2$ and this sibling group was chosen in Step 4 of the current invocation, make one recursive call $\text{MAF}(F_1, F_2 \div \{e_{a_1}, e_{p_{a_1}}, e_{a_2}\})$ and return 3 plus its return value.
 - 7.2. If $m = 2$ and this sibling group was chosen in Step 4 of a previous invocation, make one recursive call $\text{MAF}(F_1, F_2 \div \{e_{a_1}, e_{p_{a_1}}, e_{a_2}, e_{p_{a_2}}\})$ and return 4 plus its return value.
 - 7.3. If $m > 2$ and this sibling group was chosen in Step 4 of the current invocation, let a_1 and a_2 be the two entries with maximum priority in Q , ordered so that $d_{a_1} \geq d_{a_2}$. If a_2 's parent has a single sibling, this sibling is a member a_j of the current sibling group, and a_1 's parent is either a root or has a sibling that is not a member of the current sibling group, let $x = 2$; otherwise let $x = 1$. Remove a_x from Q , make one recursive call $\text{MAF}(F_1, F_2 \div \{e_{a_x}, e_{p_{a_x}}\})$, and return 2 plus its return value.
 - 7.4. If $m > 2$ and this sibling group was chosen in Step 4 of a previous invocation, delete the node a_1 with maximum priority from Q , make one recursive call $\text{MAF}(F_1, F_2 \div \{e_{a_1}, e_{p_{a_1}}\})$, and return 2 plus its return value.

Using these modifications, we obtain the following theorem.

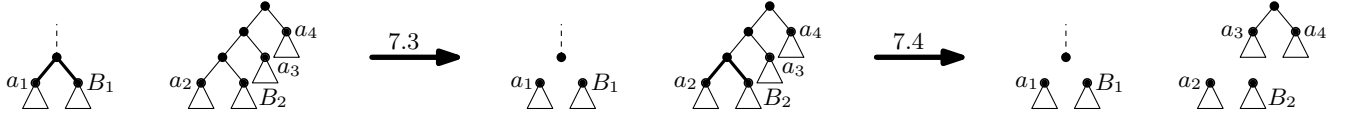
(a) $m = 2$: A single application of Case 7.1 resolves the sibling group $\{a_1, a_2\}$.



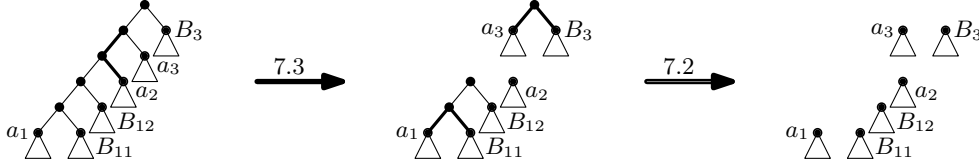
(b) $m = 4$: A single application of Case 7.3 resolves the sibling group $\{a_1, a_2, a_3, a_4\}$.



(c) The set of edges cut on the same input if we did not give preference to a_2 in Case 7.3. Since cutting edge e_{B_2} makes the sibling group agree with F_1 in this case, this would not give a 3-approximation.



(d) $m = 4$: The optimal solution needs to cut 2 edges, while our algorithm cuts 6.



(e) The sequence of cuts on the same input if we did not give preference to a_2 in Case 7.3. Note that we obtain the same forest.

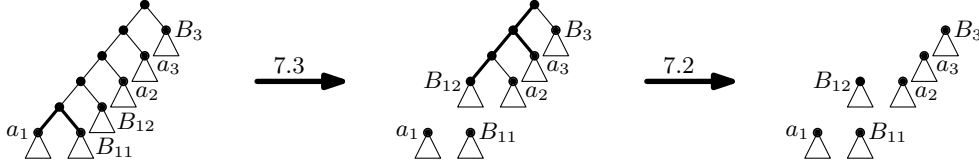


Figure S7: Illustration of the various cases of Step 7 of the approximation algorithm. Only \hat{F}_2 is shown. Edges that are cut in each step are shown in bold.

Theorem 3. Given two rooted X -trees T_1 and T_2 , a 3-approximation of $e(T_1, T_2, T_2) = d_{SPR}(T_1, T_2)$ can be computed in $O(n \log n)$ time.

Proof. We use the algorithm just described. This algorithm consists of Steps 2–5 of the FPT algorithm plus the modified Step 7 above. In addition, there is a linear-time preprocessing step for computing the initial depth estimates of all nodes of T_2 . We argued already that all executions of Step 5 take linear time in total. In Section S7, we argued that each execution of Step 2, 3 or 4 of the FPT algorithm takes constant time. In the approximation algorithm, if Step 4 chooses a new sibling group, it also needs to insert the members of the sibling group into the priority queue. This takes $O(m \log m)$ time, $O(n \log n)$ in total for all sibling groups. Each execution of Step 7 takes $O(\log n)$ time, constant time for the modifications of \hat{F}_2 it performs and $O(\log n)$ time for the $O(1)$ corresponding priority queue operations. Thus, to obtain the claimed time bound of $O(n \log n)$ for

the entire algorithm, it suffices to show that each step of the algorithm is executed $O(n)$ times.

This is easy to see for Steps 3, 5, and 7: Each execution of Step 5 reduces the number of nodes in R_t by one, the number of nodes in R_t never increases, and initially R_t contains the n leaves of T_1 . Each execution of Step 3 or 7 cuts at least one edge in F_1 or F_2 , and initially these two forests have $O(n)$ edges.

For Steps 2 and 4, we observe that they cannot be executed more often than Steps 3, 5, and 7 combined because any two executions of Step 2 or 4 have an execution of Step 3, 5 or 7 between them.

It remains to bound the approximation ratio of the algorithm. First observe that the value k' returned by the algorithm satisfies $k' \geq e(T_1, T_2, T_2)$ because the input forest F_2 of the final invocation $\text{MAF}(F_1, F_2)$ is an AF of T_1 and T_2 and the algorithm returns the number of edges cut to obtain F_2 from T_2 . To prove that $k' \leq 3e(T_1, T_2, T_2)$, let r be the number of descendant invocations of the current invocation $\text{MAF}(F_1, F_2)$, not counting the current invocation itself, and let k'' be its return value. We use induction on r to prove that $k'' \leq 3e(T_1, T_2, F_2)$ if $\text{MAF}(F_1, F_2)$ chooses a new sibling group in Step 4. We call such an invocation a *master invocation*. We also consider the last invocation of the algorithm to be a master invocation. For two master invocations without another master invocation between them, all invocations between the two invocations are *slave invocations* of the first of the two master invocations, as they manipulate the sibling group chosen in this invocation.

As a base case observe that, if $r = 0$, then $e(T_1, T_2, F_2) = 0$ and the invocation $\text{MAF}(F_1, F_2)$ returns $k'' = 0$ in Step 2. For the inductive step, consider a master invocation $\mathcal{I} = \text{MAF}(F_1, F_2)$ with $r > 0$, and let $\mathcal{I}' = \text{MAF}(F'_1, F'_2)$ be the first master invocation after \mathcal{I} . By the inductive hypothesis, \mathcal{I}' returns a value k''' such that $k''' \leq 3e(T_1, T_2, F'_2)$.

If $m = 2$ in invocation \mathcal{I} , we invoke Case 7.1, which cuts edges e_{a_1} , $e_{p_{a_1}}$, and e_{a_2} and, thus, makes the sibling group agree between F_1 and F_2 (see Figure S7(a)). This implies that $k'' = k''' + 3$. By Lemma 6, we have $e(T_1, T_2, F'_2) \leq e(T_1, T_2, F_2) - 1$. Since $k''' \leq 3e(T_1, T_2, F'_2)$, this shows that $k'' \leq 3e(T_1, T_2, F_2)$.

If $m > 2$ in invocation \mathcal{I} , this invocation applies Case 7.3, each of its slaves applies Case 7.2 or 7.4, and Case 7.2 is applied at most once. Since the sibling group $\{a_1, a_2, \dots, a_m\}$ does not agree between F_1 and F_2 , at least one edge cut in F_2 is necessary to make the sibling group agree between F_1 and F_2 . We distinguish whether one or more edge cuts are required.

If one cut suffices (Figures S7(b) and S7(c)), then there are at most two components of F_2 that contain members of the current sibling group because resolving overlaps in F_1 between q components of F_2 requires at least $q - 1$ cuts in F_2 . Consequently, exactly one component C contains at least two members of the sibling group. The existence of at least one such component follows because $m > 2$. If we had another component C' containing at least two members of the current sibling group, then at least one cut would be required in each of C and C' to make the sibling group agree between F_1 and F_2 , but we assumed that one cut suffices.

For a single cut to suffice to make the current sibling group agree between F_1 and F_2 , C must consist of a single path of nodes x_1, x_2, \dots, x_t such that, for $1 \leq j < t$, x_j has two children: x_{j+1} and a member a_{i_j} of the current sibling group; x_t has a member a_{i_t} of the current sibling group as a child, as well as a group B_{i_t} of siblings of a_{i_t} such that no member a_h of the current sibling group belongs to $F_2^{B_{i_t}}$. Thus, cutting edges $e_{a_{i_t}}$ and $e_{p_{a_{i_t}}}$ makes the sibling group agree between F_1 and F_2 . Now observe that a_{i_t} and $a_{i_{t-1}}$ are the two members of the current sibling group with the greatest depth estimates in C . If there exists another component C' of F_2 that contains a member a_h of the current sibling group, then a_h is the only such node in C' . Thus, the two maximum priority entries in Q are either a_{i_t} and $a_{i_{t-1}}$ or a_{i_t} and a_h . In both cases, invocation \mathcal{I} cuts edges $e_{a_{i_t}}$ and

$e_{p_{a_{i_t}}}$ because a_h either has no parent or its parent does not have a member of the current sibling group as a sibling, and a_{i_t} is preferred over $a_{i_{t-1}}$ by invocation \mathcal{I} because a_{i_t} 's parent does have a member of the current sibling group as its only sibling (namely $a_{i_{t-1}}$) and has a greater depth estimate than $a_{i_{t-1}}$. Thus, in \mathcal{I} 's child invocation, the current sibling group agrees between F_1 and F_2 , which implies that this child invocation is \mathcal{I}' and $k'' = k''' + 2$. Moreover, since the sibling group does not agree between F_1 and F_2 in invocation \mathcal{I} , we must have $e(T_1, T_2, F'_2) \leq e(T_1, T_2, F_2) - 1$ and, hence, $k'' = k''' + 2 \leq 3e(T_1, T_2, F'_2) + 2 \leq 3e(T_1, T_2, F_2)$.

For the remainder of the proof assume at least two cuts are necessary to make the sibling group $\{a_1, a_2, \dots, a_m\}$ agree between F_1 and F_2 (Figures S7(d) and S7(e)), and assume the members of the sibling group are ordered by their depth estimates. Let i_1, i_2, \dots, i_s be the indices such that invocation \mathcal{I} and its slaves cut edges $\{e_{a_{i_j}}, e_{p_{a_{i_j}}} \mid 1 \leq j \leq s\}$ and, for all $1 \leq j \leq s$, let B_{i_j} be the set of a_{i_j} 's siblings at the time we cut edges $e_{a_{i_j}}$ and $e_{p_{a_{i_j}}}$. Assume for now that invocation \mathcal{I} cuts

edges e_{a_1} and $e_{p_{a_1}}$. Then, for all $1 \leq j \leq s$, $F_2^{B_{i_j}}$ contains no member of the current sibling group because such a member a_h would have a greater depth estimate than a_{i_j} and hence e_{a_h} would have been cut before $e_{a_{i_j}}$. This implies that there exists an edge set E such that $F_2 \div E$ is an MAF of F_1 and F_2 and $|E \cap \{e_{a_{i_j}}, e_{B_{i_j}} \mid 1 \leq j \leq s\}| \geq s - 1$. Since cutting edges $e_{a_{i_j}}$ and $e_{B_{i_j}}$ produces the same result as cutting edges $e_{a_{i_j}}$ and $e_{p_{a_{i_j}}}$, this shows that $e(T_1, T_2, F'_2) \leq e(T_1, T_2, F_2) - (s - 1)$.

If $s \geq 3$, we have $2s \leq 3(s - 1)$ and, hence, $k'' \leq k''' + 3(s - 1) \leq 3(e(T_1, T_2, F'_2) + s - 1) \leq 3e(T_1, T_2, F_2)$. If $s < 3$, we observe that $e(T_1, T_2, F'_2) \leq e(T_1, T_2, F_2) - 2$ because the current sibling group agrees between F_1 and F'_2 and we assumed that at least two cuts are necessary in F_2 to make this sibling group agree between F_1 and F_2 . Thus, $k'' \leq k''' + 2s \leq 3e(T_1, T_2, F'_2) + 4 \leq 3e(T_1, T_2, F_2)$.

It remains to deal with the case when invocation \mathcal{I} cuts edges e_{a_2} and $e_{p_{a_2}}$. If $a_1 \notin F_2^{B_2}$, then again $F_2^{B_{i_j}}$ contains no member of the current sibling group, for all $1 \leq j \leq s$, and the same argument as above shows that $k'' \leq 3e(T_1, T_2, F_2)$. If $a_1 \in F_2^{B_2}$, then observe that either the path in F_2 between a_1 and p_{a_2} has at least two internal nodes or a_2 has at least two siblings because otherwise invocation \mathcal{I} would prefer a_1 over a_2 because a_1 has the greater depth estimate. This implies that a_2 has the same parent before and after cutting edges e_{a_1} and $e_{p_{a_1}}$, and a_1 has the same parent before and after cutting edges e_{a_2} and $e_{p_{a_2}}$. Thus, \mathcal{I} and its child invocation cut the same four edges e_{a_1} , $e_{p_{a_1}}$, e_{a_2} , and $e_{p_{a_2}}$ as would have been cut if invocation \mathcal{I} had not preferred a_2 over a_1 . The same argument as above now shows that $k'' \leq 3e(T_1, T_2, F_2)$. \square